Proceedings of the 3rd Planning to Learn Workshop (WS9) at ECAI 2010

Edited by: Brazdil, Pavel ; Bernstein, Abraham ; Kietz, Jörg-Uwe

Abstract: The task of constructing composite systems, that is systems composed of more than one part, can be seen as interdisciplinary area which builds on expertise in different domains. The aim of this workshop is to explore the possibilities of constructing such systems with the aid of Machine Learning and exploiting the know-how of Data Mining. One way of producing composite systems is by inducing the constituents and then by putting the individual parts together. For instance, a text extraction system may be composed of various subsystems, some oriented towards tagging, morphosyntactic analysis or word sense disambiguation. This may be followed by selection of informative attributes and ?nally generation of the system for the extraction of the relevant information. Machine Learning techniques may be employed in various stages of this process. The problem of constructing complex systems can thus be seen as a problem of planning to resolve multiple (possibly interacting) tasks. So, one important issue that needs to be addressed is how these multiple learning processes can be coordinated. Each task is resolved using certain ordering of operations. Meta-learning can be useful in this process. It can help us to retrieve previous solutions conceived in the past and re-use them in new settings. The aim of the workshop is to explore the possibilities of this new area, offer a forum for exchanging ideas and experience concerning the state-of-the art, permit to bring in knowledge gathered in different but related and relevant areas and outline new directions for research. It is expected that the workshop will help to create a sub-community of ML / DM researchers interested to explore these new venues to ML / DM problems and help thus to advance the research and potential for new type of ML / DM systems.

THE 19TH EUROPEAN CONFERENCE ON ARTIFICIAL INTELLIGENCE (ECAI 2010)

3rd PLANNING TO LEARN WORKSHOP WS9 AT ECAI 2010

PlanLearn-2010

August 17, 2010

Lisbon, Portugal

Edited by

Pavel Brazdil, Abraham Bernstein, Jörg-Uwe Kietz

A Brief History

The current PlanLearn-2010 workshop is 3rd one in the series.
The first one workshop, PlanLearn-2007, was associated with ECML/PKDD-07 that took place on Warsaw Poland. The invited speaker was Larry Hunter, Univ. of Colorado at Denver and Health Sciences Center who presented a talk entitled *Historical Overview of the area Planning to Learn*.
The second workshop, PlanLearn-2008, was associated with ICML/COLT/UAI that took place in Helsinki, Finland. The invited speaker was Raymond J. Mooney from the University of Texas at Austin who presented a talk on *Transfer Learning by Mapping and Revising Relational Knowledge*.
More information about PlanLearn-2010 and its predecessors can be found at
`http://www.ifi.uzh.ch/ddis/events/planlearn10/`

Motivation

The task of constructing composite systems, that is systems composed of more than one part, can be seen as interdisciplinary area which builds on expertise in different domains. The aim of this workshop is to explore the possibilities of constructing such systems with the aid of Machine Learning and exploiting the know-how of Data Mining. One way of producing composite systems is by inducing the constituents and then by putting the individual parts together.
For instance, a text extraction system may be composed of various subsystems, some oriented towards tagging, morphosyntactic analysis or word sense disambigua- tion. This may be followed by selection of informative attributes and finally generation of the system for the extraction of the relevant information. Machine Learning tech- niques may be employed in various stages of this process. The problem of constructing complex systems can thus be seen as a problem of planning to resolve multiple (possibly interacting) tasks.
So, one important issue that needs to be addressed is how these multiple learning processes can be coordinated. Each task is resolved using certain ordering of operations. Meta-learning can be useful in this process. It can help us to retrieve previous solutions conceived in the past and re-use them in new settings.
The aim of the workshop is to explore the possibilities of this new area, offer a forum for exchanging ideas and experience concerning the state-of-the art, permit to bring in knowledge gathered in different but related and relevant areas and outline new directions for research. It is expected that the workshop will help to create a sub-community of ML / DM researchers interested to explore these new venues to ML / DM problems and help thus to advance the research and potential for new type of ML / DM systems.

Main Areas Covered by the Workshop

Of particular interest are methods and proposals that address the following issues:

- Planning to construct composite systems,
- Exploitation of ontologies of tasks and methods,
- Representation of learning goals and states in learning,
- Control and coordination of learning processes,
- Recovering / adapting sequences of DM operations,
- Meta-learning and exploitation of meta-knowledge,
- Layered learning,
- Multi-task learning,
- Transfer learning,
- Multi-predicate learning (and other relevant ILP methods),
- Combining induction and abduction,
- Multi-strategy learning,
- Learning to learn.

The proceedings include 11 contributions covering many of the topics mentioned above. We thank everyone for their interest and their contribution. Besides, the program includes two invited talks:

- Michele Sebag from LRI, U. Paris-Sud will present a talk on *Monte-Carlo Tree Search: From Playing Go to Feature Selection*,
- Luc de Raedt, from Katholieke Universiteit Leuven, Belgium will present a talk on *Constraint Programming for Data Mining*

We are very happy that they accepted our invitation.

Acknowledgements

The organization of this workshop is partially supported by the European Community 7th framework program ICT-2007.4.4 under grant number 231519 *"e-Lico: An e-Laboratory for Interdisciplinary Collaborative Research in Data Mining and Data-Intensive Science"*. Gratitude is also expressed to FCT for the Pluriannual funding provided to LIAAD-INESC Porto L.A., that helped its members to organize this event.

We wish that everyone have an interesting and inspiring workshop leading to good follow-up work afterwards!

Lisbon, July 2010

Pavel Brazdil
Abraham Bernstein
Jörg-Uwe Kietz

Workshop Organization

Workshop Chairs

Pavel Brazdil (University of Porto)
Abraham Bernstein (University of Zurich)
Jörg-Uwe Kietz (University of Zurich)

ECAI Workshop Chair

Ulle Endriss (University of Amsterdam, The Netherlands)

Workshop Program Committee

Abraham Bernstein, Switzerland
Hedrik Blockeel, Belgium
Pavel Brazdil, Portugal
Christophe Giraud-Carrier, USA
Saso Dzeroski, IJS, Ljubljana
Peter Flach, Great Britain
Larry Hunter, USA
Jörg-Uwe Kietz, Switzerland
Rui Leite, Portugal
Tom Mitchell, USA (advisory role)
Oliver Ray, Great Britain
Ashwin Ram, USA
Luc de Raedt, Belgium
Carlos Soares, Portugal
Maarten van Someren, The Netherlands
Vojtech Svatek, Czech Republic
Ricardo Vilalta, USA
Filip Zelezny, Czech Republic

Technical support

Management of EasyChair: Abraham Bernstein and Jörg-Uwe Kietz

Preparation of Proceedings: Rui Leite

Table of Contents

An Overview of Intelligent Data Assistants for Data Analysis

Floarea Serban and **Jörg-Uwe Kietz** and **Abraham Bernstein**[1]

Abstract. Todays intelligent data assistants (IDA) for data analysis are focusing on how to do effective and intelligent data analysis. However this is not a trivial task since one must take into consideration all the influencing factors: on one hand data analysis in general and on the other hand the communication and interaction with data analysts. The basic approach of building an IDA, where data analysis is (1) better as well as (2) faster in the same time, is not a very rewarding criteria and does not help in designing good IDAs. Therefore this paper tries to (a) discover constructive criteria that allow us to compare existing systems and help design better IDAs and (b) review all previous IDAs based on these criteria to find out what are the problems that IDAs should solve as well as which method works best for which problem. In conclusion we try to learn from previous experiences what features should be incorporated into a new IDA that would solve the problems of todays analysts.

1 Introduction

As technology advances generating larger amounts of data becomes easier, increasing the complexity of data analysis. The process of analyzing data is not a trivial task requiring time, experience and knowledge about the existing operators. As new types of data appear and corresponding algorithms are designed to analyze them, the task of the data analyst becomes more complex since it is requiring awareness of a continuos expanding set of operators [32].

Several data analysis (DA) systems have been developed to analyze data, each of them providing a large number of operators (Clementine, SAS Enterprise Miner, RapidMiner, Weka). Even if they are constantly trying to improve the quality of the analysis, the number of operators and the data size are growing making it difficult for users to find an appropriate sequence of operators for a given task or data set. Therefore with the continuous evolution of these systems there is an essential need to provide more assistance to the users and suggest the right operators for their current task. Even if this need has already been mentioned in the literature [22], to our knowledge, there is no IDA which could successfuly solve this problem. Several attempts in developing IDAs have been made [52, 4, 21] but they were either just a proof of concept [4, 20] or they are no longer maintained [52]. Thus they are unusable for today's data analysts. Moreover they are focusing either on novice analysts or on experts and therefore cannot provide support for all users. Furthermore they are either guiding the user step by step or leaving her free to make her own decisions, so the user is either fully restricted to a fixed set of steps or left alone in a jungle of operators. Therefore finding the right sequence of operators becomes almost impossible.

It is difficult to precisely define how the best IDA should be designed. Nevertheless we can say that the basic (minimal) features of an IDA should be defined in terms of quality and the amount of time spent for the analysis . Hence an IDA is basically defined by its goals: (1) to analyze the data better with the IDA than without, given an amount of time, and (2) to do it faster with the IDA support than without, given a required level of quality. The quality of the analysis is important since an IDA should improve the quality and help the user obtain better results. In addition time is always a problem, large data sets take a long of time to be analyzed therefore the assistant should decrease this time and do the analysis faster. An IDA should definitely include these two features, but they are not sufficient to design a good system. Therefore in this paper we develop a set of constructive criteria that allow us to (a) compare existing systems and (b) help design a better IDA. Moreover we review all the previous attempts to develop IDAs based on these criteria to find out (a) which problems IDAs should solve and (b) which method works best for which problem. Furthermore, based on the comparison with existing IDAs, we present a set of features for a new IDA, called EG-IDA (explorative guidance IDA) which helps the user explore and effectively use the data analysis techniques.

The rest of the paper is organized as follows: Section 2 presents existing work on IDAs or intelligent systems developed for data analysis, makes an overview of their features and their applicability in current scenarios and highlights some of their limitations, then Section 3 introduces the desired features of a new IDA that should overcome the limitations of existing IDAs and finally conclusions and future work are described in Section 4.

2 IDAs evolution

Over the course of time scientists worked on developing systems to improve data analysis. One of the first research directions was to automate the data analysis process as well as provide more user support, therefore introducing the Statistical Expert Systems (SES) at the beginning of '80s [33, 25]. At that time the data analysis process was mainly based on statistics; statisticians were studying the relationships between variables. Systems like REX [24] tried to improve the user support for data analysis by using the knowledge from expert users and generating rules. These rules were used to help users in solving data analysis problems like linear regression. The advice was limited to the encoded expert knowledge and could be applied only on a reduced set of problems. Moreover the systems could not handle more complex questions and were restrained to a hardcoded set of answers.

[1] University of Zurich, Department of Informatics, Dynamic and Distributed Information Systems Group, Binzmühlestrasse 14, CH-8050 Zurich, Switzerland {serban|kietz|bernstein}@ifi.uzh.ch

This led to an extension of statistical expert systems, the knowledge enhancement systems (KES) which try to offer a more flexible access to statistical expertise. Such an approach is that of KENS [35], a KES for non-parametric statistics, which assists the user in solving problems and tries to improve her understanding of nonparametric statistics. The user can ask questions in natural language and the system provides a list of answers based on both human and machine expertise. The successor of KENS is NONPAREIL [35] which still focuses on nonparametric statistics but it uses hypertext links instead of searching. A similar approach is LMG [35] which assists the user in fitting linear models. In fact LMG presents to the user questions which contain hypertext buttons through which she can have access to explanations about a selected concept. The KES systems give more freedom to the users in exploring the statistical world. They also represent learning environments - the user can easily learn by questioning the system how to handle different problems from the domain (either nonparametric statistics or linear modeling).

The next evolution step was the appearance of Intelligent Exploratory Data Analysis systems. Systems like AIDE [52] or Data Desk [53] offer intelligent support for exploratory data analysis. They provide means that make the analyst's task easier by improving the user interactivity or the user support with more explanations. Finally the evolution continued with more focus on the IDAs in systems like IDEA [4], CITRUS [54], MetaL [41]. Remarkable work was done by Robert Engels who describes in his PhD thesis the User Guidance Module (UGM) [23] - a cookbook on how to do Data Mining as well as a written and also implemented (part of CITRUS) IDA. Another "written" IDA is the CRISP-DM process model [13] since it is a step-by-step data mining guide - it is considered the standard process model for Data Mining. Even if CRISP-DM is just a standard by the fact that it presents guidance on how to do data mining it constitutes itself as an IDA.

The present survey covers several different systems[2] as shown in Table 1 that can be grouped in the following categories: Statistical Expert Systems (SES), Knowledge Enhancement Systems (KES), Exploratory Data Analysis Systems (EDA), Data Analysis Systems (DAS) and Automatic Data Assistants (ADA). The comparison was done based on the published research papers for the SES and EDA since the systems are no longer available. For DA systems we were able to try and check the functionality of the current systems, Clementine, RapidMiner, Weka, SPSS, Excel, Matlab and R which facilitate access to a collection of algorithms but they offer no real decision support to inexperienced end-users.

2.1 Comparison of existing IDAs

All the enumerated systems do intelligent data analysis since they offer guidance (partially) and help the user to analyze her data. We identified seven possible dimensions on which the systems could be compared:

- Support for modeling a **single-step** from the KDD process vs. **multi-step** KDD process
  We compare systems which help the user to model a single step from the KDD process by guiding her on how to use a specific operator, how to choose the right parameters for it, to systems which provide support for multi-step processes - they assist the user in the selection and application of available techniques at each step of the DM process. But an IDA should include both these dimensions: the user needs information on configuring a specific operation as well as building the sequence of steps from the KDD process.
- **Graphical editing** vs. **automatic generation**
  Graphical editing refers to enabling the user to draw the process manually - choose the operators, set the right inputs/outputs and parameters. While automatic generation provides the user with a set of workflows (or at least one) that she needs to execute in order to solve the data mining task. The system automatically sets all the inputs/outputs and the parameters for each operator. This is useful for users that don't have too much experience with the data mining operators. Based on the data and a description of their task they get a set of possible scenarios of solving a problem. Both of the criteria are recommended for an IDA since the users have different experience and knowledge and they need different help.
- **Re-use past experiences** vs. **generation from scratch**
  The system saves all the produced cases and records the ones which have succeeded or have failed to solve the given task. Reusing past cases improves the generation of new better workflows since the system reuses only the similar cases or parts of the cases that were successful. Moreover it saves time because the system doesn't have to generate each workflow from scratch and also helps avoiding the repetition of mistakes. Reusing past cases is definitely an asset for an IDA since it can improve recommendations and should be considered when implementing such an IDA. An IDA should definitely include the first feature, being able to reuse cases can improve the assistants recommendations.
- **Task decomposition** vs. **plain plan**
  Task decomposition structures and breaks down the complexity of a KDD task. It originates from the field of knowledge acquisition where it was used to describe and specify complex tasks [21]. Task descriptions can be reused thus decreasing the development time and simplifying the process of decomposing a KDD task. Also the task role is to transform the initial problem with certain features into the goal problem with additional features [12].
- **Design support** vs. **explanations for result/output**
  By design support we refer to the help and advice the system provides during the design of the data mining process. The user can easily find information about operators, he is given input or hints on how to solve problems or errors. Opposed to design support the explanations help the user interpret the results by suggesting different methods (e.g., graphs). Explanation includes the support for the interpretation of intermediate and final results as well as the capability to explain the reasoning and decisions of the system. Both features should be included in IDAs since they help the user exploring and discovering the operators.
- **Experimental** vs. **analytical** approach
  Experimental systems enable the user to execute the workflow, she or the system creates, and visualize the results. Contrary to experimental systems analytical ones are based on rules learned either from experts or from data characteristics. But both are required for an IDA - one enables the user to execute operators and the other one can give recommendations on how and when to use the operators.
- **Data mining experience level**
  The systems can be used by users with a certain DM knowledge level; we have systems which consider naive users, others which can be used only by experienced users, by experts or by users with any level of knowledge. Also for some systems the intended user is not specified or the systemy can't be used since they are just a proof of concept.

[2] For systems without have name we considered the name of the main author.

Category	System name	KDD single step support	KDD multi-step support	Graphical workflow editing	Automatic workflow generation	Re-use past experiences	Task decomposition	Design support	Explanations	Experimental	Analytical	DM experience level	References
SES	REX	++	−	−−	−−	−−	−−	−−	++	−	++	naive	[24]
	SPRINGEX	++	−	−−	−−	−−	−−	−−	+	−	++	experienced	[49]
	STATISTICAL NAVIGATOR	++	−	−−	−−	−−	−−	−−	++	−	++	experienced	[49]
	MLT Consultant	++	−	−−	−−	−−	−−	−−	++	++	++	all	[50]
KES	KENS	++	−	−−	−−	−−	−−	−−	+	−	++	experienced	[34, 35]
	LMG	++	−	−−	−−	−−	−−	−−	+	−	++	all	
	GLIMPSE	++	−	−−	−−	−−	−−	−−	++	−	++	experienced	[55, 56]
EDAS	AIDE	+	+	−	+	+	+	−	+	++	−	experienced	[52]
	Data Desk	0	0	−	−	−	−	−	−	++	−	all	[53]
DAS	SPSS	0	0	−	−	−	−	−	+	++	−	experienced	
	Clementine	0	0	++	−	−	−	++	++	++	−	experienced	
	SAS Enterprise Miner	0	0	++	−	−	−	++	++	++	−	experienced	[11]
	Weka	0	0	++	−	−	−	+	−	++	−	experienced	[30]
	RapidMiner 5.0	0	0	++	−	−	−	++	+	++	−	experienced	[46]
	KNIME	0	0	++	−	−	−	++	+	++	−	experienced	[5]
	Orange	0	0	++	−	−	−	++	+	++	−	experienced	[17]
	R	0	0	−	−	−	−	−	−	++	−	experienced	[40]
	MATLAB	0	0	−	−	−	−	+	−	++	−	experienced	[44]
	Excel	0	0	−	−	−	−	−	−	++	−	all	[43]
	Data Plot	0	0	−	−	−	−	++	−	++	−	unspecified	[37]
	GESCONDA	0	0	−	−	−	−	+	−	++	−	unspecified	[28, 27]
ADAS	IDEA	−	++	−	++	−	−	−	−	++	−	unspecified	[4]
	CITRUS	−	++	−	++	++	++	−	++	++	−	all	[21, 22]
	Záková	−	++	−	++	−	−	−	−	++	−	unspecified	[58, 57]
	KDDVM	−	++	−	++	−	−	−	−	+	−	experienced	[18, 19]
CBRS	METAL (DMA)	++	−	−	−	++	−	−	−	−	++	experienced	[41, 29]
	Mining Mart	0	0	−	−	++	−	−	−	++	−	experienced	[48, 42]
	Charest	0	0	−	−	++	−	−	−	++	−	all	[14, 15]
Other	CRISP-DM	+	++	−−	−−	−−	++	−−	−	−−	++		[13]
	IDM	++	−	−	−	−	−	++	+	+	+	unspecified	[6]
	MULREG	++	−	−	−	−	−	−	+	++	−	all	[20]

++ = well supported (a main feature of the system)
\+ = supported
0 = neutral, the system can do it but there is no assistance
− = not present but integrable
−− = not possible

Table 1: List of IDAs by category

2.1.1 Modeling a single-step from the KDD process vs. multi-step

Initially the systems provided support for modeling a single step from the KDD process, like SES and KES do. REX focuses on linear regression advice as opposed to the commercially available SES which have a broader application domain and can provide advice on several problems. Hence Springex handles bivariate, multivariate and non-parametric statistics and Statistical Navigator covers even more statistical techniques: multivariate causal analysis, scaling and classification, exploratory data analysis, etc.. The system gives advice based on the information provided by the user combined with the rules from the knowledge base. KES are more flexible from the point of view on how they present the advice - the user is free to question the systems and presents all the answers to the user. GLIMPSE even suspends when information is missing, and suggests actions to find that missing information. A more complex approach is that of MLT-Consultant [50] which provides advice on how to use a specific algorithm from the Machine Learning Toolbox (MLT). The advice is based on a knowledge base containing rules extracted from real-world tasks achieved by the domain experts and also from the interaction with the ML algorithm developers. As contribution the Consultant-2 provides support for preprocessing data as well as suggesting new methods after the one applied produced unsatisfactory results. Thus a step from the KDD process is seen not as a single-step but as a cyclic process where the user can reapply other algorithms if she is not satisfied with the current results. Moreover it is one of the first attempts to use meta-learning - they tried to suggest the appropriate ML tool based on the task description, the data characteristics and the output format. A similar view is the one of MULREG which recommends certain techniques for linear regression basing its advice on metadata (the measurement level) and on properties of the data themselves (parameters of their distribution). Also MetaL uses the notion of *meta-learning* to advise users which induction algorithm to choose for a particular data-mining task [38]. One of the outcomes of the project was a Data Mining Advisor (DMA) [29] based on meta-learning that gives users support with model selection. IDM has a knowledge module which contains meta-knowledge about the data mining methods, and it is used to determine which algorithm should be executed for a current problem. GESCONDA provides several tools for recommending a proper way to face the analysis in order to extract more useful knowledge like method suggestion, parameter setting, attribute/variable meta-knowledge management, etc. Other work was done by the StatLog project [45] which has investigated which induction algorithms to use given particular circumstances. This approach is further explored by [8, 26] which use meta-rules drawn from experimental studies, to help predict the applicability of different algorithms; the rules consider measurable characteristics of the data (e.g., number of examples, number of attributes, number of classes, kurtosis, etc.). [10] present a framework which generates a ranking of classification algorithms based on instance based learning and meta-learning on accuracy and time results.

DAS are neutral from this point of view - the user is free to choose an operator, set the parameters she wants to use and execute it. A totally opposite direction is the one of IDEA system which provides users with systematic enumerations of valid DM processes and rankings by different criteria. The enumeration is done based on the characteristics of the input data and of the desired mining result as well as on an operator ontology which specifies preconditions and effects for each operator. However the system doesn't in fact support the user through the steps of the DM process it just enumerates the steps.

The shift from single-step to multi-step started with the introduction of the CRISP-DM standard and the CITRUS system which helps the user through all the phases of the KDD process. Current DAS enable the user to design and execute multi-step KDD processes, but this becomes hard when the processes have a large number of operators.

2.1.2 Graphical editing vs. automatic generation

SPSS Clementine (SPSS Modeler nowadays), SAS Enterprise Miner [3] and RapidMiner 5.0 [4] enable the user to draw workflows manually as opposed to ADA systems which generate them automatically based on planning techniques. IDEA uses straightforward search to automatically output the valid processes. The user can select the plan by choosing a ranking method (accuracy, speed, etc.). As opposed to IDEA, the approach in AIDE differs by the way the user and the machine interact: AIDE offers a step by step guidance based on the script planner and the user's decisions. This is suitable for exploratory statistics but it is not suitable for domains where the algorithms run for an extensive period of time.

2.1.3 Re-use past experiences vs. generation from scratch

The Mining Mart project proposed a *case-based reasoning* approach that enables both automatization of preprocessing and reusability of defined preprocessing cases for data mining applications [42]. The best-practice cases of preprocessing chains developed by experienced users are stored and then reused to create new data-mining processes therefore saving time and costs [59]. Moreover Mining Mart includes *cases with self-adapting operators* by using multi-strategy learning [42]. MetaL project developed also a case-based system which combines knowledge and data to advise users which induction algorithm to choose for a particular data-mining task [38]. Recent work in the area [16, 14, 15] tries to combine case based reasoning with ontologies to create a hybrid DM assistant. Their CBR implementation is based on the extension of the classical meta-learning problem from mapping datasets to models, to mapping DM problems to DM cases and on a complementary utility-oriented similarity measure. Also AIDE tries to find similarities between structures to be able to reuse previous results as well as previous decisions [2].

2.1.4 Task decomposition vs. plain plan

Task-oriented user guidance was proposed by Engels and implemented in CITRUS. It guides the users by breaking down the complexity of a typical KDD task and supports him in selecting and using several machine learning techniques. The user guidance module from CITRUS offers assistance in the selection and application of available techniques, interpretation and evaluation of results. It focuses on support and not on automation. AIDE uses hierarchical problem decomposition techniques therefore goals can be decomposed into several subgoals. Problem decomposition and abstraction constitute helpful features for the exploration [52]. CRISP-DM follows a hierarchical process model, having a set of tasks at four levels of abstraction: phase, generic task, specialized task and process instance. The DM process consists of 6 phases, each of them comprise several generic tasks which cover all the possible data mining situations. The specialized tasks describe how the actions from the generic tasks should be accomplished in certain situations. The last level, process instance, represents a record of actions and results of a specific data mining operation.

2.1.5 Design support vs. explanations for result/output

The last version of RapidMiner 5.0 has more user support by introducing the *quick fixes* and the *meta-data propagation* features. Each time the user draws an operator, if it is not connected properly to other operators or some of the input or output fields have incorrect types then the system suggests a set of quick fixes - it gives advice on how the problems could be solved. After loading the data the system extracts and propagates the meta-data such that at any point it can make recommendations. This option can be found in Clementine and in SAS Enterprise Miner (propagation of information) as well. These features represent the support offered to the user during design time. Help buttons are available for each page in IDM. SPSS has more support than other systems for providing *explanations*, the help menu provides extensive information about methods, algorithms, etc. even with examples illustrating the explained feature. Additionally SPSS has coaches that take you step-by-step through the process of interpreting results or deciding which statistical analyses to choose by providing helpful examples - *learning by example* is a very useful feature. SAS Enterprise Miner has integrated debugging and runtime statistics.
REX [24] helps the user in *interpreting intermediate and final results* and also gives useful instructions about statistical concepts. Springex has a primitive 'why' explanation facility which consists of a list of rules that have succeeded together with the conditions that have been asserted. But the knowledge it is unclear - it does not provide explanation of technical terms, is superficial and incomplete. On the contrary Statistical Navigator uses an expert system with help and explanation facilities. Additionally it has extensive reporting capabilities, including a short description of each technique and references to the literature and statistical packages that implement the technique. KENS and LMG provide explanations for concepts but they don't handle interpretation of results or explanation of the reasoning. Contrarily GLIMPSE advises the user on possible interpretations of the output from the statistics package. Moreover it is built on top of a logic-based system shell, APES [31] which offers rule-based explanations - how, why and why not. Another approach is that of IDM which can interpret data by using several statistical measures but it has limited explanation facilities.

2.1.6 Experimental vs. analytical approach

In current systems users can execute workflows, thus the systems provide experimental support (i.e., Clementine, RapidMiner, Weka, etc.) as opposed to SES where the actions are suggested based on the expert knowledge or to MetaL which uses meta-learning. SES and KES are modules developed for existing statistical packages to provide guidance to the users. Therefore the execution is done by those packages. IDM stands between experimental and analytic since it can contain implementation of its own algorithms as well from other data mining systems.

2.1.7 DM experience level

REX can be used not only by expert statisticians but also by naive users. GLIMPSE and KENS are focusing on users with a certain level of knowledge, opposed to LMG which adapts to user expertise. The

[3] Last version of Enterprise Miner is 6.1.You can find a description of the new features here: http://support.sas.com/documentation/onlinedoc/miner/.
[4] RapidI provides RapiMiner: http://rapid-i.com/content/view/181/190/. Last version is 5.0.

inexperienced users can easily feel lost in SPRINGEX and Statistical Navigator since they offer a large amount of knowledge. MULREG is designed for both statisticians and non-statisticians, as well as MLT Consultant and CITRUS which can be used by any domain expert (with any level of DM knowledge). Current DM systems like Weka, Clementine, RapidMiner, etc. have a large set of operators. Even if they have explanations and help facilities it is not trivial for a naive user to solve DM tasks, therefore users need to have experience in using the tool and getting used to the DM domain.

2.1.8 Other features

Visual exploration is a necessity for a system which does data analysis. Therefore most of the existing systems (RapidMiner 5.0, Clementine, DataDesk) as well as exploratory IDAs (AIDE) offer different visualization facilities like icons, interactive graphs, interactive tables, etc. As suggested in [53], *interactivity* is the key to a good EDA, thus it is a desired feature for an IDA as well. Data Desk offers different interactive features like drag and drop, pop-up-menus, hype view menus which provide guidance through potential analyses. Also introduces the concept of extensive user interaction by using links to all graphs and tables.

Hierarchical organization of operators in an ontology, like in IDEA [4], KDDVM [18] and Záková[57], provides several benefits like the inheritance as well as the possibility to introduce abstract operators. An ontology is used as well by [15] to assist novice data miners by storing rules which encode recommendations for DM operators.

Improved navigation plays an important role in user performance in AIDE [1, 3]. Amant has identified navigation functions which lead to improvements in almost any system for statistical data analysis. AIDE includes three types of navigation operations : user actions (go/step, back, forward, any, history, copy/paste, delete, replay), system actions (go/jump, look, refine) and communication actions (overview type, justify,, history, zoom, magnify, filter, landmark, paths).

Ranking of workflows consists an important feature of IDEA - it uses speed and accuracy to provide the user with effective rankings. This facilitates user's decision in the selection of one process that will solve her task. Other work has been done in the field of ranking of classification algorithms based on the accuracy and performance on similar data sets [51] or using different ranking algorithms like average ranks, success rate ratio and significant wins [9].

Preventing serious errors is a feature implemented as well in Clementine and RapidMiner 5.0 by detecting errors based on metadata propagation.

User preferences - IDM stores the history of each user, including user preference and creates profiles for them. Also AIDE [2] incorporates user preferences into the analysis, it records user's decisions and allows her to go backward and forward and to modify her choices. Even MULREG stores and preserves history of the previous models and data structures such that the user can reuse them across different days. KNIME offers a nice feature to the user: a favorite node view where users can manage their favorite nodes, most frequently used and last used.

2.2 Limitations of existing systems

All the systems described in the previous sections are different ways of doing intelligent data analysis, but none of them is a full-IDA system such that it assists the user in doing data analysis.

SES and KES focus on a limited area of DA, thus they work with a limited number of operators. The systems from these two categories were developed during the 80s and are focusing on the needs at that time. Today's needs are much more broad, data analysis includes nowadays a lot of new methods and algorithms for analyzing data. Moreover the size of the data has increased substantially.

Current DA systems as well as SES and KES have less support for automatic workflow design. Even if the user can manually draw the workflow (e.g. RapidMiner, Clementine) for large workflows it can be quite time consuming therefore it would be advisable to generate workflows automatically starting from the features of the input data and the description of the task.

Using most of the systems requires previous knowledge and experience either with the DM operators or with the statistics techniques. Some of the existing systems are developed for naive users (e.g, REX, [16]), to help them solve their tasks easily but then they become too easy for an expert user - they are not helping but restricting her. An IDA should take into consideration all kinds of users and try to integrate facilities for all of them.

Most of the IDAs lack the possibility to reuse similar cases. In DAS users can reopen their previous workflows and try to find out which is the most convenient for the selected task and data but this decision is difficult to be made by a user. It is more a task for the computer - based on algorithms it can figure out which case is most adequate for the current task and data.

Also the explanation support is rather limited and quite superficial. Most of the systems don't have support for interpreting results - this is an advanced feature that would be helpful for the users. Anyway even basic features like description of existing operators are either missing or too scarce.

They either restrict the user in following a set of steps (question-answering systems like REX, KENS) or let him free without any orientation steps (DAS). Restriction is good for naive users - they are usually lost in a jungle of operators or algorithms and it is hard for them to figure out a path to follow. On the other hand expert users prefer to have independence, they know the path to follow but sometimes they have doubts and therefore they need to check with the system if their decisions are correct.

Most of current IDAs don't take user's actions and history into consideration. Users are unique, they use different operators and produce different workflows. It would be helpful to take user's history into consideration and try to learn from it.The user interface could adapt and present to the user only the most used operators, restricting the selection field.

3 Desired features of the future EG-IDA

The IDA should offer explorative guidance to the user - it should help the user discover the Data Mining operators and help him improve the quality of the data mining task and easily solve his task in a shorter time. Besides the features described in Section 2.1, based on the previous work, it is recommended for the assistant to combine the techniques from both mixed-initiative user interfaces [39, 36] and mixed-initiative planning. As argued in [36] the mixed initiative interaction is an essential feature of systems which perform complex tasks as our IDA is doing. Moreover integrating an automated system together with direct manipulation (user's actions) can generate a more natural collaboration between humans and computers. We think that the IDA should at least integrate all the following services to help users increase their productivity:

- **Automatic workflows** - the system automatically generates the

sequence of operators which could solve the user's task. That is helpful for naive users that have little knowledge about the algorithms and methods but also for experts since they can check their knowledge and even find new ways of solving a task.
- **Execution of one or more workflows** - the IDA can find several ways in solving the task so the user should be able to execute alternative workflows at the same time.
- **Step by step execution of a workflow** - the user may want to see how an operator works and what it produces, also she should be able to pause, stop, resume the execution of such a workflow.
- **Ranking of workflows** - for some task the IDA may provide a high number of possible workflows and hence it is difficult for the user to choose one of them. An approach would be to use ranking as suggested in [4] and present to the user the workflows by different criteria : speed, length, accuracy, etc.
- **Reusing workflows** based on their data and goal similarity - it involves case based reasoning - storing all the workflows in a case-base and each time a new workflow has to be generated it will first try to retrieve similar workflows (the similarity will be computed based on the data the user provides and her goal description).
- **Enter goals/hints** and generate proposals of how to reach them
- **Detects errors**, and propose how to repair them
- **Allow the user design his workflows** by using levels of abstraction (in operators, tasks and methods) - the user can use abstractions for operators, methods and tasks and the system can recommend which of the basic instances would better fit the workflow the user has drawn

Moreover it should be compliant to the mixed-initiative interfaces where the user can do anything manually, though :

- At any time she can delegate tasks to the system - the user is not restricted by the system, the user can independently execute her tasks but she can ask the system to solve specific tasks. Also if the task description is not clear enough the system can ask for clarification or can ask questions to the user such that it gets a more specific description.
- In the background the system infers the user's goals and helps to reach them - the system watches over the user's intentions, retrieves all her actions and decisions and based on this information infers what the user wants to achieve and finds possible ways of doing it.
- At any time she can ask the system "what to do next" - at some point the user might not be able to decide what to do next or she has too many decisions that she could make, then the user asks the system to tell her what to do next, therefore the system and the user collaborate and together solve tasks.
- For any decision or step the system can provide explanations - the system offers explanatory information about its decisions and reasoning, tells the user why a specific next step was executed and not another one, thus the user can understand all the system's actions and can learn from them.
- Similar to a spell-checker, the system watches over the correctness and can propose corrections - the system can detect errors and propose fixes, it checks the user's actions and if it finds errors it signals them to the user.

A similar approach is the one described in [47] where the system uses **balanced cooperation** - the modeling task can be done by the user or by a tool of the system. Moreover the user controls the modeling process and guides the learning. There is a synergy between the user and the system, both contribute to model building but the system supports the user, does not take decisions, on the contrary the system is guided by the user. The AIDE system is a mixed initiative system which makes suggestions to the user as well as responds to user guidance about what to do next. The role of mixed-initiative interfaces in intelligence analysis is clearly stated in [7]. Collaboration between the analyst and the system is essential to a better analysis, so the system represents an assistant and not only a tool. As described in [36] the system and the user work as a team, assisting and helping each other but they can also execute some of the tasks independently when required. An important issue is which of the two participants should have the control and when - so when should the system interrupt the user? Mainly the user should have control of the execution and when the system is asked to solve a task it can ask the user more information. Also it would be helpful if the system could provide suggestions to the user when the user is doing nothing or when the user seems lost in the user interface. But the system can help the user intrinsically in any situation without taking control of the execution - for example when the user wants to draw an workflow the system can automatically connect the corresponding nodes or give suggestions for connections.

4 Conclusion

Most of the current IDAs focus on data analysis itself, they offer the users a set of operators to analyze their data. But since the size of the data increased significantly in the last years as well as new types of data have appeared (image, multimedia, etc.) it is hard for users to use the existing tools. There are several attempts to enrich the data analysis systems with guidance but are either too restrictive - offer to the users a set of fixed steps that need to be made, or they provide insufficient guiding. Thus the users have to make decisions based on their own experience and for naive users is not a trivial task.

This paper tried to analyze the IDAs history, identify the problems of existing IDAs and learn from both failure and success. Therefore we introduced a set of metrics based on the existing/proposed implementations of IDAs and looked at their advantages as well as limitations. Moreover we proposed a set of recommended features for designing a new generation of IDAs. The new generation IDA combines techniques from both mixed-initiative interfaces and mixed-initiative planning, therefore it combines automation with user's decisions.

As next steps we intend to implement a new IDA based on the proposed metrics and features. We are convinced that a new IDA is going to fill the gap between analysts/users and todays data analysis systems and also make a faster and better analysis.

ACKNOWLEDGEMENTS

This work is partially supported by the European Community 7^{th} framework program ICT-2007.4.4 under grant number 231519 "e-Lico: An e-Laboratory for Interdisciplinary Collaborative Research in Data Mining and Data-Intensive Science".

REFERENCES

[1] R. Amant, 'Navigation for data analysis systems', *Advances in Intelligent Data Analysis Reasoning about Data*, 101–109, (1997).
[2] R.S. Amant and P.R. Cohen, 'Preliminary system design for an EDA assistant', in *Preliminary Papers of the Fifth International Workshop on Artificial Intelligence and Statistics*, volume 3. Citeseer, (1995).
[3] R.S. Amant and P.R. Cohen, 'Interaction with a mixed-initiative system for exploratory data analysis', *Knowledge-Based Systems*, **10**(5), 265–273, (1998).

[4] Abraham Bernstein, Foster Provost, and Shawndra Hill, 'Towards Intelligent Assistance for a Data Mining Process: An Ontology-based Approach for Cost-sensitive Classification', *IEEE Transactions on Knowledge and Data Engineering*, **17**(4), 503–518, (April 2005).

[5] M.R. Berthold, N. Cebron, F. Dill, T.R. Gabriel, T. Kötter, T. Meinl, P. Ohl, C. Sieb, K. Thiel, and B. Wiswedel, 'KNIME: The Konstanz information miner', *Data Analysis, Machine Learning and Applications*, 319–326, (2008).

[6] R. Bose and V. Sugumaran, 'Application of intelligent agent technology for managerial data analysis and mining', *ACM SIGMIS Database*, **30**(1), 94, (1999).

[7] L.K. Branting, 'The Role of Mixed-Initiative Agent Interfaces in Intelligence Analysis: Extended Abstract', (2005).

[8] P. Brazdil, J. Gama, and B. Henery, 'Characterizing the applicability of classification algorithms using meta-level learning', in *Machine Learning: ECML-94*, pp. 83–102. Springer, (1994).

[9] P. Brazdil and C. Soares, 'A comparison of ranking methods for classification algorithm selection', *Machine Learning: ECML 2000*, 63–75, (2000).

[10] P.B. Brazdil, C. Soares, and J.P. Da Costa, 'Ranking learning algorithms: Using IBL and meta-learning on accuracy and time results', *Machine Learning*, **50**(3), 251–277, (2003).

[11] P.B. Cerrito, *Introduction to Data Mining Using SAS Enterprise Miner*, SAS Publishing, 2007.

[12] B. Chandrasekaran, T.R. Johnson, and J.W. Smith, 'Task-structure analysis for knowledge modeling', *Communications of the ACM*, **35**(9), 124–137, (1992).

[13] P. Chapman, J. Clinton, T. Khabaza, T. Reinartz, and R. Wirth, 'The CRISP-DM process model', *The CRIP–DM Consortium*, **310**, (1999).

[14] M. Charest, S. Delisle, O. Cervantes, and Y. Shen, 'Intelligent Data Mining Assistance via CBR and Ontologies', in *Proceedings of the 17th International Conference on Database and Expert Systems Applications (DEXA'06)*, (2006).

[15] M. Charest, S. Delisle, O. Cervantes, and Y. Shen, 'Bridging the gap between data mining and decision support: A case-based reasoning and ontology approach', *Intelligent Data Analysis*, **12**(2), 211–236, (2008).

[16] M. Charest, S. Delisle, and O. Cervants. Design considerations for a CBR-based intelligent data mining assistant, 2006.

[17] J. Demšar, B. Zupan, G. Leban, and T. Curk, 'Orange: From experimental machine learning to interactive data mining', *Knowledge Discovery in Databases: PKDD 2004*, 537–539, (2004).

[18] C. Diamantini, D. Potena, and E. Storti, 'Kddonto: An ontology for discovery and composition of kdd algorithms', *THIRD GENERATION DATA MINING: TOWARDS SERVICE-ORIENTED*, 13, (2009).

[19] C. Diamantini, D. Potena, and E. Storti, 'Ontology-Driven KDD Process Composition', *Advances in Intelligent Data Analysis VIII*, 285–296, (2009).

[20] W. DuMouchel, 'The structure, design principles, and strategies of Mulreg', *Annals of Mathematics and Artificial Intelligence*, **2**(1), 117–134, (1990).

[21] R. Engels, 'Planning tasks for knowledge discovery in databases; performing task-oriented user-guidance', in *Proceedings of the International Conference on Knowledge Discovery & Data Mining AAAI-Press, Portland, OR*, pp. 170–175, (1996).

[22] R. Engels, G. Lindner, and R. Studer, 'A guided tour through the data mining jungle', in *Proceedings of the 3nd International Conference on Knowledge Discovery in Databases. Newport Beach, CA*, (1997).

[23] Engels, R., *Component-based user guidance in knowledge discovery and data mining*, IOS Press, 1999.

[24] W.A. Gale, 'REX review', in *Artificial intelligence and statistics*, pp. 173–227. Addison-Wesley Longman Publishing Co., Inc., (1986).

[25] W.A. Gale, AT, and T Bell Laboratories., *Artificial intelligence and statistics*, Addison-Wesley Pub. Co., 1986.

[26] J. Gama and P. Brazdil, 'Characterization of classification algorithms', *Progress in Artificial Intelligence*, 189–200, (1995).

[27] K. Gibert, X. Flores, I. Rodríguez-Roda, and M. Sànchez-Marrè, 'Knowledge discovery in environmental data bases using GESCONDA', in *Transactions of the 2nd Biennial Meeting of the International Environmental Modelling and Software Society*, volume 1, pp. 51–56.

[28] K. Gibert, M. Sānchez-Marrč, and I. Rodriguez-Roda, 'GESCONDA: An intelligent data analysis system for knowledge discovery and management in environmental databases', *Environmental Modelling & Software*, **21**(1), 115–120, (2006).

[29] C. Giraud-Carrier, 'The data mining advisor: meta-learning at the service of practitioners', in *Machine Learning and Applications, 2005. Proceedings. Fourth International Conference on*, p. 7, (2005).

[30] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I.H. Witten, 'The WEKA data mining software: An update', *ACM SIGKDD Explorations Newsletter*, **11**(1), 10–18, (2009).

[31] P. Hammond and M. Sergot, 'Augmented PROLOG for Expert Systems', *Logic Based Systems Ltd*, (1984).

[32] D.J. Hand, 'Intelligent data analysis: issues and opportunities', in *Advances in Intelligent Data Analysis. Reasoning about Data: Second International Symposium, IDA-97, London, UK, August 1997. Proceedings*, p. 1. Springer.

[33] DJ Hand, 'Statistical expert systems: design', *The Statistician*, **33**(4), 351–369, (1984).

[34] DJ Hand, 'A statistical knowledge enhancement system', *Journal of the Royal Statistical Society. Series A (General)*, **150**(4), 334–345, (1987).

[35] Hand, D. J., 'Practical experience in developing statistical knowledge enhancement systems', *Annals of Mathematics and Artificial Intelligence*, **2**(1), 197–208, (1990).

[36] M.A. Hearst, 'Mixed-initiative interaction', *IEEE Intelligent systems*, **14**(5), 14–23, (1999).

[37] A. Heckert and JJ Filliben, 'DATAPLOT Reference Manual', *NIST Handbook*, **148**, (2003).

[38] M. Hilario and A. Kalousis, 'Fusion of meta-knowledge and meta-data for case-based model selection', *Principles of Data Mining and Knowledge Discovery*, 180–191.

[39] E. Horvitz, 'Principles of mixed-initiative user interfaces', in *Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit*, pp. 159–166. ACM, (1999).

[40] R. Ihaka and R. Gentleman, 'R: A language for data analysis and graphics', *Journal of computational and graphical statistics*, **5**(3), 299–314, (1996).

[41] A. Kalousis, J. Gama, and M. Hilario, 'On data and algorithms: Understanding inductive performance.', *Machine Learning Journal, Special Issue on Meta-Learning, , 54(3), 275-312*, **54**(3), 275–312, (2004).

[42] Kietz, J.-U. and Vaduva, A. and Zücker, R., 'Mining mart: combining case-based-reasoning and multi-strategy learning into a framework to reuse KDD-application', in *Proceedings of the fifth International Workshop on Multistrategy Learning (MSL2000). Guimares, Portugal*, volume 311, (2000).

[43] D.M. Levine, M.L. Berenson, and D. Stephan, *Statistics for managers using Microsoft Excel*, Prentice Hall, 1999.

[44] I. MathWorks, 'Matlab', *The MathWorks, Natick, MA*, (2004).

[45] D. Michie, D.J. Spiegelhalter, C.C. Taylor, and J. Campbell, 'Machine learning, neural and statistical classification', (1994).

[46] Ingo Mierswa, Michael Wurst, Ralf Klinkenberg, Martin Scholz, and Timm Euler, 'Yale: Rapid prototyping for complex data mining tasks', in *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 935–940. ACM, (2006).

[47] K. Morik, 'Balanced cooperative modeling', *Machine Learning*, **11**(2), 217–235, (1993).

[48] K. Morik and M. Scholz, 'The MiningMart Approach to Knowledge Discovery in Databases', in *Intelligent Technologies for Information Analysis*, eds., Ning Zhong and Jiming Liu, 47 – 65, Springer, (2004).

[49] J.F.M. Raes, 'Inside two commercially available statistical expert systems', *Statistics and Computing*, **2**(2), 55–62, (1992).

[50] D. Sleeman, M. Rissakis, S. Craw, N. Graner, and S. Sharma, 'Consultant-2: Pre-and post-processing of machine learning applications.', (1995).

[51] C. Soares and P. Brazdil, 'Zoomed ranking: Selection of classification algorithms based on relevant performance information', *Principles of Data Mining and Knowledge Discovery*, 160–181, (2000).

[52] R. St. Amant and P.R. Cohen, 'Intelligent support for exploratory data analysis', *Journal of Computational and Graphical Statistics*, **7**(4), 545–558, (1998).

[53] M. Theus, 'Exploratory data analysis with data desk', *Computational Statistics*, **13**(1), 101–116, (1998).

[54] R. Wirth, C. Shearer, U. Grimmer, T. Reinartz, J. Schlösser, C. Breitner, R. Engels, and G. Lindner, 'Towards process-oriented tool support for knowledge discovery in databases', *Principles of Data Mining and Knowledge Discovery*, 243–253, (1997).

[55] D.E. Wolstenholme and C.M. O'Brien, 'GLIMPSE-a statistical adventure', in *Proceedings of the 10th International Joint Conference on Ar-*

tificial Intelligence (IJCAI 87, Milan, 23-28 August 1987), volume 1, pp. 596–599. Citeseer, (1987).

[56] D.E. Wolstenholme, C.M. O'Brien, and J.A. Nelder, 'GLIMPSE: a knowledge-based front end for statistical analysis', *Knowledge-Based Systems*, **1**(3), 173–178, (1988).

[57] M. Záková, P. Kremen, F. Zelezný, and N. Lavrac, 'Using Ontological Reasoning and Planning for Data Mining Workflow Composition', in *ECML 2008 Workshop on Third Generation Data Mining: Towards Service-oriented Knowledge Discovery*. Citeseer, (2008).

[58] M. Záková, V. Podpecan, F. Zelezný, and N. Lavrac, 'Advancing data mining workflow construction: A framework and cases using the orange toolkit', *ECML PKDD 2009*, 39, (2009).

[59] Zücker, R. and Kietz, J.-U. and Vaduva, A., 'Mining Mart: Metadata-Driven Preprocessing', in *Proceedings of the ECML/PKDD Workshop on Database Support for KDD*. Citeseer, (2001).

eProPlan: A Tool to Model Automatic Generation of Data Mining Workflows

— Extended Abstract for System Demo —

Jörg-Uwe Kietz and **Floarea Serban** and **Abraham Bernstein**[1]

Abstract. This paper introduces the first ontological modeling environment for planning Knowledge Discovery (KDD) workflows. We use ontological reasoning combined with AI planning techniques to automatically generate workflows for solving Data Mining (DM) problems. The KDD researchers can easily model not only their DM and preprocessing operators but also their DM tasks, that are used to guide the workflow generation.

1 Introduction

Current DM-Suites support the user only with the manual creation of KDD workflows, but this is time consuming and requires experience and knowledge about the DM operators as well as the DM-Suites. Over the course of time several people have tried to build systems that can automate this process[1, 2, 11]. These approaches have shown that AI planning is a way of supporting the user with automatically generated workflows. But modeling for planning is difficult as well since one has to describe the DM domain in terms of operations that can be applied. To our knowledge none of these systems contain a model of the operators of a state-of-the-art DM-Suites nor are they publicly available.

In this paper we present eProPlan[2], the first ontology-based Integrated Development Environment (IDE) for planning applications. Together with the DMWF-DMO (Data Mining Work Flow-Ontology) and over 100 modeled RapidMiner operators it is the first publicly available planner for planning DM workflows. We also defined a programming interface (IDA-API), such that generated workflows can be delivered directly to DM-Suites like RapidMiner or general Service-Workflow engines like Taverna [6].

In Section 2 we present the architecture of our system, Section 3 describes the steps and audience of the demonstration, we conclude with a short discussion of innovations of eProPlan over related work in Section 4.

2 eProPlan Architecture

The system comprises several Protégé 4 [9] plugins, that allow the users to describe and solve DM problems. By using the public-domain ontology-editor Protégé 4 as the base environment we exploit the advantages of an ontology as a formal model for the domain knowledge. But instead of over-using the ontological inferences for planning (as done in [3, 12]) we decided to extend the ontological formalism with the main components of a plan, namely operator conditions & effects for classic planning and tasks & methods for Hierarchical Task Network (HTN) planning [10].

DMO is our DM-Ontology. It consists of two main parts: The DMWF [7] which contains IO-objects, operators, goals, tasks and methods as well as the decomposition of tasks into methods and operators. The DMWF is used by the AI-planner. The DMOP [4] focuses on the operators' features and is used by the probabilistic-planner to optimize the plans generated by the AI-planner.

eProPlanI is our reasoner plugin, i.e. the interface of our reasoner & planner for Protégé. It combines ontological reasoning and HTN-planning. Other plugins (like eProPlanP) rely on it since they retrieve and display useful information about the DM process (operators and their applicability). **eProPlanO** is an editor for operators. Operators have conditions and effects expressed in an extended SWRL language [5] as shown in [7]. Our customized tab displays for each operator its conditions and effects (both current and inherited), only current conditions and effects can be edited. We extended the SWRL editor from Protégé 4 and built our own editor that has a syntax checker and a variable binding checker.

eProPlanM provides the support for HTN modeling as a task/method editor, users can define their own task/method decompositions. Methods and tasks' modeling – consists of three steps and is managed by a customized tab with three class views. The first view presents a tree with the decomposition of tasks into methods, methods into tasks and operators classes. The user can easily add/ delete new tasks and methods as well as specify their decomposition into substeps. The next view displays the conditions and contributions for methods. The third view shows the method bindings for a selected method.

eProPlanG represents a DM-task specification editor. It enables the user to specify the input data as well as describe the DM-task she wants to solve in terms of main goals and optional subgoals the planner has to reach.

eProPlanP is the support for workflow execution and visualization. It has mainly two functionalities: provides the user a step-by-step planner – it displays applicable operators for a data set and can apply them, and shows all the plans for a specific DM task.

IDA-API is the programming interface to the reasoner & planner used to build Intelligent Discovery Assistants (IDA) based on the services of the planner. Within the e-Lico project Rapid-I is currently using this API to integrate the planner into the leading open-source DM-Suite RapidMiner, such that RapidMiner users can get automatically generated DM workflows to execute them there.

[1] University of Zurich, Department of Informatics, Dynamic and Distributed Information Systems Group, Binzmühlestrasse 14, CH-8050 Zurich, Switzerland {kietz|serban|bernstein}@ifi.uzh.ch

[2] eProPlan, DMWF-DMO and RapidMiner-operator ontology can be downloaded from http://www.e-lico.eu/eProPlan. The IDA-API and its integration into RapidMiner will be available soon. For a first impression this site also contains screen-shoots and preliminary demo videos.

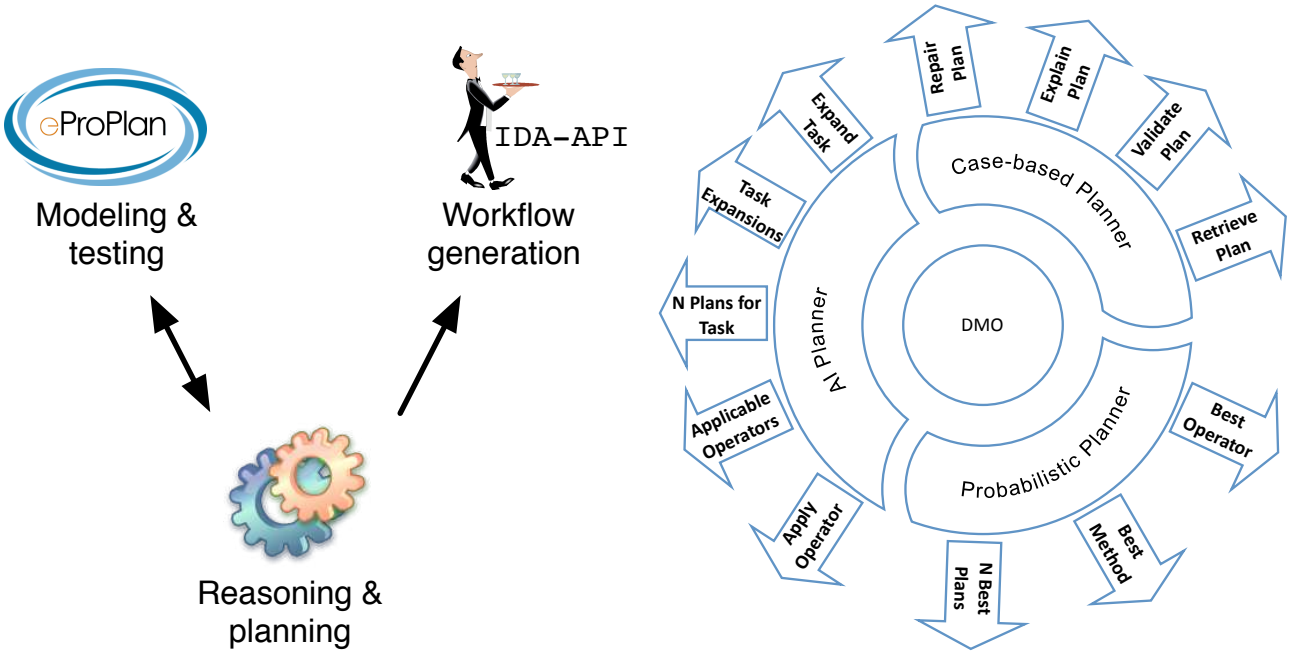


Figure 1: (a) eProPlan architecture (b) The services of the planner

The complete system will have the services displayed in Figure 1b which are developed as part of the e-Lico project. Currently the AI planner's services are available, the other two parts are under development.

3 Demonstration

The demonstration will cover the whole life-cycle of DM-workflow planning from modeling data sets, preprocessing- and DM-operators, DM-goals and task/method decompositions, via testing the model in eProPlan by entering specific goals and getting the DMWF-metadata description of concrete data sets from a data analysis service, to the generation of complete DM-workflows within eProPlan and within RapidMiner, where the generated workflows can be executed.

The demo is of interest for the KDD researchers as well as for the KDD practitioners. Researchers may want to model their own operators to be used in generated workflows. An advanced usage scenario is that researchers model their own task/method decompositions, e.g. to let the planner generate systematic experiments to be executed by a DM-Suite. DM-Suite developers may be interested in the services the system can add to their suite. Practitioners can get an impression of how future systems can support them in their daily work of doing DM projects. An extended description of the system and how it is used for auto-experimentation is available in [8].

4 Conclusions

There are many attempts at automating the generation of KDD workflows: Žáková et. al [12] automatically generate workflows using a knowledge ontology and a planning algorithm based on the Fast-Forward system. However, they only return the shortest workflow with the smallest number of processing steps – no alternatives are generated. IDEA [1], the Intelligent Discovery Assistant (IDA), provides users with systematic enumerations of valid DM processes. It is based on an ontology of DM operators that guides the workflow composition and contains heuristics for the ranking of different alternatives. CITRUS [11] consists of an IDA which offers user-guidance through mostly a manual process of building the workflows. It uses planning for plan decomposition and refinement.

Similar to IDEA our system also provides many plans. We believe that this is a necessary condition as the system may not know all the tradeoffs between operators and/or the user's desiderata. Also, as far as we know, ePropPlan is the first and only KDD workflow planner that combines this ability with an integrated Planning Development Environment, can be easily integrated into existing DM suites through its IDA-API, as it is publicly available.

Acknowledgements: This work is supported by the European Community 7^{th} framework ICT-2007.4.4 (No 231519) "e-Lico: An e-Laboratory for Interdisciplinary Collaborative Research in Data Mining and Data-Intensive Science".

REFERENCES

[1] Abraham Bernstein, Foster Provost, and Shawndra Hill, 'Towards Intelligent Assistance for a Data Mining Process: An Ontology-based Approach for Cost-sensitive Classification', *IEEE Transactions on Knowledge and Data Engineering*, **17**(4), 503–518, (2005).

[2] M. Charest, S. Delisle, O. Cervantes, and Y. Shen, 'Bridging the gap between data mining and decision support: A case-based reasoning and ontology approach', *Intelligent Data Analysis*, **12**(2), 211–236, (2008).

[3] Claudia Diamantini, Domenico Potena, and Emanuele Storti, 'KD-DONTO: An Ontology for Discovery and Composition of KDD Algorithms', in *Service-oriented Knowledge Discovery (SoKD-09) Workshop at ECML/PKDD09*, (2009).

[4] Melanie Hilario, Alexandros Kalousis, Phong Nguyen, and Adam Woznica, 'A data mining ontology for algorithm selection and meta-learning', in *Service-oriented Knowledge Discovery (SoKD-09) Workshop at ECML/PKDD09*, (2009).

[5] I. Horrocks, P.F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean, *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*, http://www.w3.org/Submission/SWRL/, 2004.

[6] D. Hull, K. Wolstencroft, R. Stevens, C. Goble, M.R. Pocock, P. Li, and T. Oinn, 'Taverna: a tool for building and running workflows of services', *Nucleic acids research*, **34**(Web Server issue), W729, (2006).

[7] J.-U. Kietz, F. Serban, A. Bernstein, and S. Fischer, 'Towards cooperative planning of data mining workflows', in *Service-oriented Knowledge Discovery (SoKD-09) Workshop at ECML/PKDD09*, (2009).
[8] J.-U. Kietz, F. Serban, A. Bernstein, and S. Fischer, 'Data mining workflow templates for intelligent discovery assistance and auto-experimentation', Technical report, University of Zürich, (06 2010). Available at `http://www.e-lico.eu/public/SoKD10.pdf`.
[9] H. Knublauch, R.W. Fergerson, N.F. Noy, and M.A. Musen, 'The Protégé OWL plugin: An open development environment for semantic web applications', *Lecture notes in computer science*, 229–243, (2004).
[10] D. Nau, T.-C. Au, O. Ilghami, U. Kuter, W. Murdock, D. Wu, and F.Yaman., 'SHOP2: An HTN planning system.', *JAIR*, **20**, 379–404, (2003).
[11] R. Wirth, C. Shearer, U. Grimmer, T. Reinartz, J. Schlösser, C. Breitner, R. Engels, and G. Lindner, 'Towards process-oriented tool support for knowledge discovery in databases', *Principles of Data Mining and Knowledge Discovery*, 243–253, (1997).
[12] M. Žáková, V. Podpečan, F. Železný, and N. Lavrač, 'Advancing data mining workflow construction: A framework and cases using the orange toolkit', in *Service-oriented Knowledge Discovery (SoKD-09) Workshop at ECML/PKDD09*, (2009).

Improving the Execution of KDD Workflows Generated by AI Planners

Susana Fernández, Rubén Súarez, Tomás de la Rosa, Javier Ortiz, Fernando Fernández, Daniel Borrajo[1] and **David Manzano** [2]

Abstract.

PDM is a distributed architecture for automating data mining (DM) and knowledge discovery processes (KDD) based on Artificial Intelligence (AI) Planning. A user easily defines a DM task through a graphical interface specifying the dataset, the DM goals and constraints, and the operations that could be used within the DM process. Then, the tool automatically obtains all the possible models that solve the task by combining the different KDD actions. The models are obtained by the execution of workflows in a DM engine. In turn these workflows are automatically generated using an state-of-the-art AI planner. Since the number of potential KDD workflows that solve a DM task can be huge, PDM allows the user to set up some quality criteria (accuracy, execution time, ...) to prune and rank the space of workflows. These criteria can be used when planning to guide the search process towards good workflows. This requires to model the effects that each KDD action has on the criteria. The first versions of the PDM Tool included estimations made by experts, and then by a machine learning module that improves them through the analysis of execution of the generated worksflows in different datasets. This paper presents our current work on the PDM Tool for improving the estimations on those values, based on a more fine grained analysis of results.

1 Introduction

Data Mining (DM) and Knowledge Discovery in Databases (KDD) is a very dynamic research and development area that is progressing constantly. Recently, researchers are defining the third generation of DM and KDD systems. The first generation of DM systems support a single algorithm or a small collection of algorithms that are designed to mine attribute-valued data. Second generation systems are characterized by supporting high performance interfaces to databases and data warehouses and by providing increased scalability and functionality. And the emerging third generation systems should be able to mine distributed and highly heterogeneous data found across the network of computer systems and integrate efficiently with operational data/knowledge management and DM systems. This implies, among other things, the implementation of DM and KDD tools to enable the construction of KDD workflows (representing potentially repeatable sequences of DM and data integration steps) [18]. Such tools should be built on the basis of standard languages and available state-of-the-art technology.

[1] Universidad Carlos III de Madrid, Leganés, Spain, email: susana.fernandez@uc3m.es

[2] Ericsson Research Spain, Madrid, Spain, email:david.manzano.macho@ericsson.com

At a high level, there are four main elements to define in the KDD process: the training data (obtained by selecting, pre-processing, and transforming the initial dataset), the model representation formalism, the learning algorithm, and how to evaluate the model. The number of combinations of those four elements is huge, since there are many different techniques appropriate for each phase, all of them with different parameter settings, which can be applied in different orders. Thus, the KDD process is sometimes seen as an expert process where DM engineers transform original data, execute different mining operators, evaluate the obtained models, and repeat this process until they are satisfied. The complexity of this combinatorial process suggests using automated approaches that are able to generate potential useful workflows and then execute them appropriately. Precisely, Automated Planning (AP) technology has become mature enough to be useful in applications that require selecting and sequencing actions [11], as it is the case of generating KDD workflows.

PDM is a tool for automatic planning of KDD workflows based on these ideas [8]. PDM describes KDD operations in terms of AP by defining a planning domain in PDDL (Planning Domain Definition Language) that is considered as the standard language in the planning community [9]. It uses another standard, the DM standard language PMML [12], to describe KDD tasks. It receives as input a KDD task, specified in a PMML file. The PMML file is automatically translated into a planning problem described in PDDL. So, any state-of-the art planner can be used to generate a plan (or plans), i.e. the sequence of KDD actions that should be executed over the initial dataset to obtain the final model. Each plan is translated into a KDD workflow and it is executed by a machine learning engine. In our case, we employ one of the most used DM tools, WEKA [21]. In WEKA, workflows are described as files with a specific format, KFML, and datasets are described as ARFF (Attribute-Relation File Format) files. The results of the KDD process can be evaluated, and new plans may be requested to the planning system. An important characteristic of PDM is that it is a distributed architecture where each module can be executed in a different host. Thus, in combination to the use of standard languages, it makes it a modular architecture, so it is possible to substitute any of its components. For instance, we could change the DM tool or the planner. We would only have to adapt the translator of plans, to deal with the input requirements of the new DM tool.

PDM represents each common KDD operator as a planning action in the PDDL domain with its preconditions and effects. The effects include estimations on the variations of the desired mining results, as execution time, accuracy and errors, or comprehensibility. In a first approach, these variations were initially set to a value estimated by an expert. But, when compared with real values obtained by executing WEKA over different datasets, we saw that those estimated val-

ues differ from the real ones. So, we enhanced the PDM architecture by integrating machine learning techniques to improve the KDD by planning process. However, the learning was performed considering the whole KDD process, i.e. PDM obtained the models and the total mininig results for each training DM task without distinguishing between the particular effects due to each KDD action. In our current work that we describe in this paper, we are separating the execution of pre-processing actions from classification/regression actions, so that we can learn better predictive models of execution of KDD actions. In parallel, we are also improving the features used for learning the models.

The paper is distributed in the following way. Section 2 presents the overall PDM architecture. Section 3 the learning component as it is currently implemented, i.e. considering the whole KDD process. Section 4 describes some initial experiments on the original PDM architecture. Section 5 explains current work on the PDM tool. Section 6 presents the related work. And, the last section draws the conclusions and suggests future work.

2 The PDM Tool

This section summarizes the PDM Tool. More details of the architecture can be found in [8]. Figure 1 shows the PDM architecture. The PDM Tool is composed of four modules: Client, Control, Datamining and Planner; each one may be executed in a different computer connected through a network. We have used the Java RMI (Remote Method Invocation) technology that enables communication between different programs running JVM's (Java Virtual Machine). The planner incorporated in the architecture is SAYPHI [6] and the DM Tool is WEKA [21]. However other DM tools could have been used. Also, any other planner that supports fluents and optimization metrics may be used too.

The *Client* module offers a graphical interface that provides access to all the application functionalities. Using the interface, a user generates a PMML file from a high level description of the DM task. Then, the *Client* module sends the PMML description to the *Control* module. At the end of the execution, it receives the results in a file.

The *Control* module interconnects all modules. It performs some translations in order to provide the data to the rest of modules in the correct format and update the learned values in the PDDL problem file. The translations needed are: from PMML to a PDDL problem, `PMML2PDDL`; and from a PDDL plan to KFML, `Plan2KFML`. The input to the module is the DM task coded in the PMML file provided by the *Client* module, the dataset and a file with planning information provided by experts. First, the `PMML2PDDL` translator generates the PDDL problem file from the PMML file with the information provided by experts. Then, the module checks if there are learned information for the planning domain. If so, the PDDL problem file is updated with the learned information; otherwise the module does not modify the PDDL problem file and continues the execution using the expert knowledge.

Then, the planner is executed to solve the translated problem. The returned set of plans is translated to several KFML files. Finally, the DM Tool executes every plan in KFML format. The *result* is a compressed file containing a set of directories, one for each plan. Each directory contains the model generated by the DM Tool, the statistics related to the evaluation of the model, the plan generated by the planner, and the corresponding KDD workflow in KFML. Once the module has the information provided by the execution of all plans, it may perform the learning task in order to obtain more accurate values for the PDDL problem files for future executions.

The *Datamining* module permits the execution of DM tasks in the WEKA DM Tool through Knowledge Flow plans. It can obtain the model output and the statistics generated as a result of the Knowledge Flow execution. The inclusion or removal of ARFF files are managed by the user through the options offered in the user interface. The input to the module is a KFML file and the output is a compressed file that contains the model generated and the statistics related to the evaluation of the model.

The *Planning* module receives the PDDL problem file generated by the *Control* module and uses the PDDL domain file, which has been previously defined for the PDM architecture. The PDDL domain contains the generic actions that can be executed by a DM tool. Each action specifies:

- Arguments: the parameters assigned to the generic action.
- Preconditions: the facts that must be achieved previous to the action execution. For instance, a model needs to be generated by a learning algorithm previous to the model evaluation
- Effects: the facts achieved with the action execution
- Costs: the estimated variation of the desired mining results, as the accuracy or execution time. Action costs are part of the action effects and are computed as a function of available features of the dataset and constant values given by the expert.

Figure 2 shows the PDDL action used for representing the DM task that learns a model with the training data. In this example, the *exec-time* function depends on the number of dataset attributes, the number of instances and the constant `model-instance-exec-time`. This constant is particular to each model received as an action argument. Thus, adjusting these constant values an expert can reproduce the different estimated durations of DM tasks. Moreover, variations on other functions, (i.e., *unreadability* and *percentage-incorrect*) are also reproducible with its particular constant values. The objective of the PDM learning component, explained in the next section, is to learn these values based on the experience of WEKA knowledge flows executions.

After solving the PDDL problem, the *Planner* module returns a set of plans in XML format ready for the conversion to a KFML format. Currently, planning tasks are solved by the SAYPHI planner [6], but the architecture could use any other planner that supports fluents, conditional effects and metrics. We have used SAYPHI because it: i) supports an extensive subset of PDDL; and ii) incorporates several search algorithms able to deal with quality metrics.

3 The Learning Component

As defined above, the PDM architecture uses expert knowledge to define some important planning information, like the time required to build an specific model, or the estimated accuracy of the resulting model. Initially, these values are defined by an expert. However, those estimations can be far from correct values, since they are hard to define. Also, it can become difficult to provide those values under all possible alternative uses of the techniques, different orders in the execution of preprocessing techniques and the different domains.

The goal of the learning component is to automatically acquire all those estimations from the experience of previous data mining processes. The data flow of this component is described in Figure 3. The main steps of this flow are:

1. Gathering Data Mining Results: the goal is to gather data mining experience from previous data mining processes. All the information is stored in an ARFF file. For a given data mining process, the following information is stored:

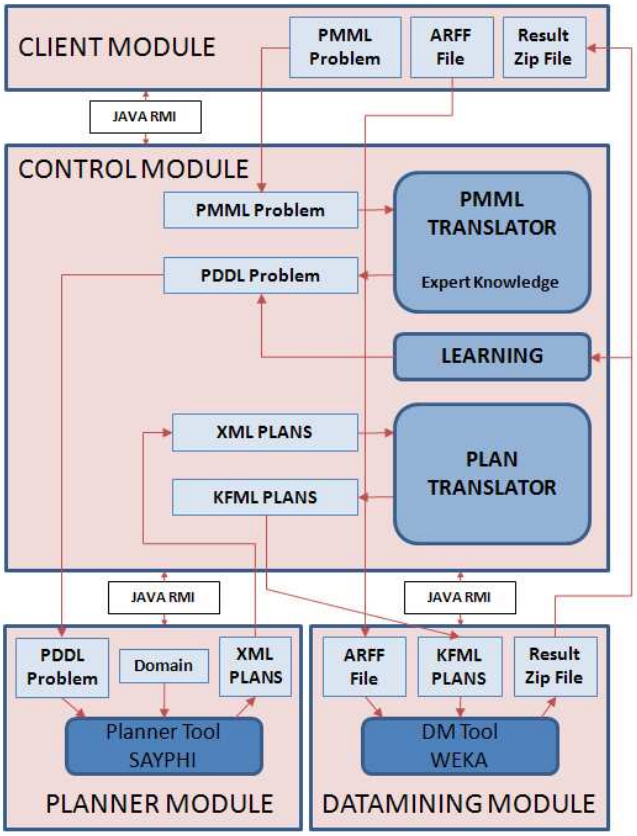


Figure 1. Overview of the PDM architecture.

```
(:action train-classification
  :parameters (?m - Model ?n - ModelName ?d - DataSet ?fi - FieldName ?t - TestMode)
  :precondition
     (and
       (implements ?m classification ?n)
       (is-field ?fi ?d)
       (is-class-field ?fi)
       (dataDictionaryDataField-otype ?fi categorical)
       (eval-on ?d ?t)
       (task-model-ready ?d))
  :effect
     (and
       (is-classification-model ?d ?n ?fi)
       (has-model-of classification)
       (not (preprocess-on ?d))
       (not (task-model-ready ?d))
       (increase (exec-time)
          (* (* (model-instance-exec-time ?n)
             (* (train-datasize-ratio ?t) (thousandsofInstances)))
             (* (dataDictionaryNumberOfFields) (dataDictionaryNumberOfFields))))
       (increase (unreadability) (model-instance-unreadability ?n))
       (increase (percentage-incorrect)
          (* (model-instance-percentage-incorrect ?n) (CorrectionFactor)))))
```

Figure 2. An example of an action in the PDDL domain file used in PDM.

- Information about the dataset: number of instances of the dataset, number of attributes of the dataset, number of continuous attributes, etc.
- Information about the model to build: the type of model (association, clustering, general regression, neural network, tree, etc.), the algorithm used to learn the model (RBFNetwork, J48, etc.), the type of function (classification, regression, clustering), the learning parameters, etc.
- Information about the results obtained: type of evaluation (split, cross validation, training set), time to build the model, accuracy, mean squared error, etc.
- The plan that represents the data mining workflow, and that has

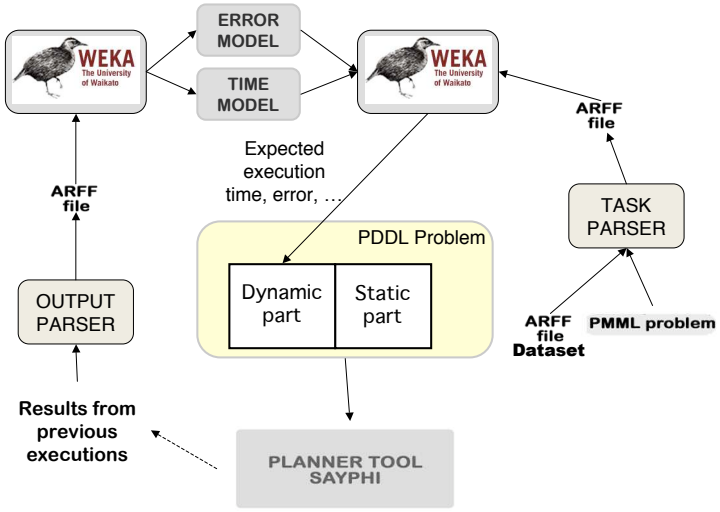


Figure 3. Learning Flow in the PDM Architecture.

been executed to obtain the model

2. Model generation: the information obtained in the previous step is used to learn prediction models. The functions to learn are time, accuracy and SME (in Figure 3, error and time models). These models can be generated with the WEKA tool, as shown in the figure.
3. Given a new dataset, a model type, a function, and a learning algorithm, and using the models generated in the previous step, we obtain a prediction of the learning time, accuracy and SME that will be obtained if we perform a new data mining process with such dataset, model type, function and learning algorithm. These estimations are included in the PDDL problem file, working out the value of the corresponding constant from the action cost formula. The updated values are then used when planning new data mining processes. Figure 4 shows an example of how the fluents of the dynamic part of the PDDL problem file are updated. In the figure, the exec-time of `treemodel1` (i.e., a specific tree model corresponding to a learning algorithm with its parameters) is updated, among others.

The updated values in the example are not final estimations, but factors used in the computation of real estimations, as defined in the action cost formula of the `train-classification` operator of the PDDL domain file (see Figure 2). For instance, when we define `(= (model-instance-exec-time treemodel1) 0.001)`, we do not mean that the execution time of learning a tree is 0.001, but that such time is computed as a function of the number of instances and attributes, weighted somehow with the learned factor 0.001.

There are two ways to update the PDDL problem file with these estimations: off-line and on-line. Off-line updates require to obtain information of many data mining processes, use the execution information to build the models, and employ these models to update the PDDL problem file, which will stay fixed in the future. On-line updates assume that, while new data-mining processes are executed, new learning examples are obtained, so the models can be dynamically updated, and the PMML problem file is continuously updated. We will only show results for off-line updates in the following section.

4 Experiments

This section presents the experimental evaluation for the original learning-component of the PDM Tool. The experiments were designed to assess the level of improvement the current implementation of the PDM Tool (i.e. considering the whole KDD process and the original features) can achieve based on the experience acquired from working with past datasets. For the evaluation, we have used twenty datasets from the UCI KDD Archive [3]. For each dataset we built a PMML file with two filters (i.e., attribute selection and discretization) and six classification algorithms (i.e., J48, IBK, Simple-Logistics, Multi-layer Perceptron, RBF-Network and SMO). We kept WEKA default parameters for both filters and algorithms. In addition, the PDDL domain considers three evaluation methods: training-dataset, split and cross-validation; also with WEKA default parameters. As a result of different combinations, we got 72 different knowledge flows for each dataset. We focus this evaluation on planning for minimizing the execution time. In order to avoid bias of arbitrarily selected values for execution time factors (fluents representing the estimated values), we decided to use equal values for the DM learning algorithms. Thus, this is the baseline for comparison that represents no expertise in the selection of these values.

The regression models for cost (time) prediction are the results of the M5Rules algorithm [10]. The reason for selecting this particular algorithm is the understandability of the output, which allows us to

[3] http://archive.ics.uci.edu/ml/datasets.html

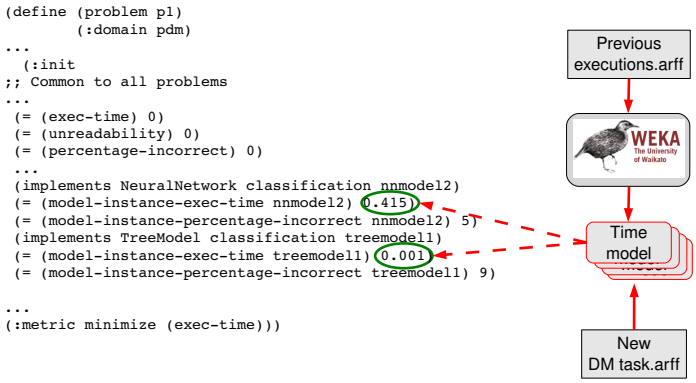


Figure 4. Example of how the expected DM results values are updated.

check the adequacy of the model. We have used the WEKA implementation of M5Rules.[4] The model is queried with dataset attributes described in section 3, to obtain a time prediction for the execution time of each algorithm. Figure 5 shows an example of a rule in one of the learned models. Examining the models obtained so far we have observed that the learning algorithm, followed by the number of attributes are the most significant attributes for time prediction (as expected). The size of the rule sets is usually small (two or three rules).

We expect that the learning configuration will make estimations closer to the real execution time. Since we do not impose particular constraints in the PMML specification, the learning configuration will return the same number of plans (72) but in a different order due to the updated values. Therefore, the evaluation can be defined as the comparison of the ranking of plans generated by the planner and the real ranking.

With this experimental setup we have performed an off-line performance evaluation. The objective is to evaluate the performance of PDM that uses the learning component, after completely training the tool. We follow the leave-one-out strategy. For each evaluated dataset, we train the tool with the 19 other datasets.

For the evaluation we follow a ranking evaluation in the spirit of [19]. We establish the plan ranking for the set of plans $\{\pi_1, \ldots, \pi_n\}$ (with $n = 72$ for this evaluation), using the plan cost function $c(\pi)$. Thus, plans are sorted in increasing order of cost (i.e., estimated execution time) where $c(\pi_i) \leq, \ldots, \leq c(\pi_j)$. The real ranking is computed from the WEKA execution of the plans/knowledge-flows. If we define as $t(\pi)$ the time for the plan execution, the real ranking is computed from the relation $t(\pi_k) \leq , \ldots, \leq t(\pi_l)$. As a result, the ranking position $R(\pi, f)$ is a function of a plan π and the function f for sorting all plans (f can be either t for real ranking order or c for planner ranking order). For instance, $R(\pi_1, c) = 10$ means that for plan π_1, there are 9 plans that have lower estimated time. For any two plans π_1 and π_2, if $R(\pi_1, c) < R(\pi_2, c)$, we consider an order switch if $R(\pi_1, t) > R(\pi_2, t)$. The interpretation over an order switch is the following: we consider an error if a plan is considered a better choice than another one by the planner, but in the real execution (with WEKA) it is actually the opposite. For the evaluation, we compute the following two measures:

- *Single ranking order switches* (Δ): It counts how many pairs (π_i, π_j) are ordered incorrectly when comparing estimated times against real times.
- *Weighted ranking order switches* ($W\Delta$): It multiplies each incorrectly ordered pair by the ranking distance as a measure of the error being committed.

These measures are normalized within the [0,1] interval, where 0 represents the perfect reverse ranking (all possible ordering errors) and 1 corresponds to the perfect ranking (no ordering errors). A random ordering criterion would obtain an average ranking score of 0.5.

4.1 Off-line Performance Results

Table 1 shows the result for the leave-one-out strategy. Some datasets for the no-learning configuration get values over the random ranking. This is mainly due to the influence that the evaluation methods have in the total execution time. For instance, if one uses the same algorithm, it is expected that a 10-fold cross-validation will last longer than a single training with a dataset split. The learning configuration improved the Δ measure in 18 out of 20 datasets, raising 11 percentage points in the average measure. The $W\Delta$ measure was improved in the same 18 datasets as well. The 14 percentage points of difference show that errors committed in order switches are less relevant with the learning configuration. Two datasets, *magicgamma* and *waveform-5000*, worsen both measures after learning. We have no explanation yet for this behaviour.

5 Current Work

This approach can be improved in two directions. The first one relates to the fact that the metric for which we learn the regression model is extracted from the execution of the whole plan, independently of the individual KDD actions that were executed, and its prediction is only used for the main DM task (classification/regression). Thus, we do not differentiate between using a couple of pre-processing techniques and not using them, and the resulting predictive model is a bit coarse. The second improvement we found out with the previous experiments relates to the input features for learning the predictive model. Therefore, our current goals are to have a more controlled

[4] http://weka.sourceforge.net/doc/weka/classifiers/rules/M5Rules.html

```
Rule: 3
time =
  0.0038 * number-instances
  + 0.163 * number-attributes
  + 61.1754 * model-instance=GRMODEL2,SVMMODEL1,NNMODEL1,TREEMODEL2,LAZYMODEL1,NNMODEL3
  - 4.3018
```

Figure 5. Example of a rule in the M5Rules time prediction model.

	No-Learning		With Learning	
Dataset	Δ	$W\Delta$	Δ	$W\Delta$
arrhytmia	0.60	0.66	0.67	0.75
car	0.53	0.58	0.72	0.82
credit-a	0.49	0.52	0.59	0.65
diabetes	0.46	0.48	0.65	0.73
glass	0.51	0.54	0.69	0.78
haberman	0.42	0.43	0.67	0.76
hepatitis	0.41	0.41	0.52	0.55
hypothyroid	0.61	0.69	0.69	0.78
iris	0.48	0.51	0.69	0.79
kr-vs-kp	0.57	0.62	0.65	0.72
magicgamma	0.62	0.70	0.59	0.65
mfeat-fourier	0.59	0.65	0.79	0.89
mushroom	0.58	0.64	0.65	0.72
nursery	0.63	0.70	0.70	0.78
optdigits	0.61	0.68	0.71	0.82
page-blocks	0.63	0.70	0.71	0.8
waveform-5000	0.62	0.69	0.60	0.65
wine	0.47	0.52	0.69	0.79
yeast	0.58	0.64	0.61	0.65
zoo	0.44	0.45	0.51	0.55
Average	0.54	0.59	0.65	0.73

Table 1. Measures for ranking order switches between estimated and real execution time in the evaluated datasets.

execution of the plans, and the use of a better set of features when learning the criteria models. Both goals aim at improving the learning step of those models, so that we have better predictive models of the behaviour of KDD actions in different datasets. In relation to the first goal, we are changing the execution of the whole plan (KDD workflow), followed by an evaluation of the results, by an execution step by step of the plan. We differentiate between "plan actions" (the actions in the domain model used by the planner) and "execution actions" (specific KDD tasks that need to be performed to consider a plan action executed). Once the planner has generated a set of plans, each of the resulting plans is divided into its plan actions. Every plan action is passed to an execution module that performs three steps: map each plan action into one or more execution actions, execute each of these execution actions and obtain its results (e.g. execution time, accuracy). The results of applying the corresponding execution actions are aggregated to compose the corresponding plan action cost. This controlled execution environment allows PDM to establish how actions are executed, not repeating actions that have already been executed in previous plans (for example loading a dataset, or applying the same sequence of filters), and providing fine-grained information about action costs.

In relation to the second goal, we have incorporated more information about the datasets, such as the meta-learning characteristics described in [5]. These features are added to the previous ones that were used to learn the model of the corresponding plan action cost estimation (execution time, accuracy, ... models). In plan cost estimation this current approach has two main advantages over previous work: a better characterization of preprocessing actions based in the dataset chracteristics where they are applied, and a characterization of an algorithm expected time and accuracy, independently from previous preprocessing actions. Examples of new features are:

- Mean standard deviation of numeric features
- Average coefficient of variation of numeric features
- Average skewness of numeric features
- Average kurtosis of numeric features
- Average normalized entropy of nominal features.
- Average of mutual information of class and attributes of nominal features
- Equivalent number of attributes of nominal features
- Noise-signal ratio of nominal features

Currently, global meta-characteristics are considered, because the representation of the domain is propositional. We are also considering posing the learning problem in the relational setting, so that we can also consider features relevant to individual attributes.

6 Related Work

The work presented in this paper can be framed into three different fields. First, it is a tool based on AI planning for assisting data mining tasks. There have been already several works on automating the data mining process through planning techniques [1, 2, 17]. The closest to our work is the second one where authors present the Intelligent Discovery Assistant (IDA). It lists potentially useful DM operations, as well as a ranking of those. They represent common DM operations in an ontology instead of in a traditional planning domain. The ontology represents a hierarchy of operations in a planning style, including preconditions, effects and costs. IDA implements its own search algorithm using that ontology and it returns a list of all possible solutions by finding all plans that achieve the goals, but without using any state-of-the-art planner. Similarly to us, each DM operator includes estimations of the effects on each goal, as accuracy or execution time, and the returned plans try to optimize them. They can dynamically include new DM operators with their corresponding estimations that influence the ranking, but they cannot automatically change the estimated values as our learning module does. Also, since they use their "ad hc" planning system, they cannot benefit from the advances on this field, as we can do by using the PDDL standard.

Second, the learning module of the PDM tool applies machine learning techniques for improving future data mining episodes. One could see it as a form of meta-learning [3]. Meta-learning studies methods that exploit meta-knowledge to obtain efficient models and solutions by adapting machine learning techniques. Apart from the techniques necessary to build meta-learning systems, it involves addressing some tasks that are also important for our work. For example, what the typical properties of datasets (or meta-features) are that

have the strongest impact on learning [15] or how to profit from the repetitive use of a predictive model over similar tasks [20]. The description of a dataset in terms of its meta-features appeared for the first time within the framework of the European StatLog project [16], whose purpose was the comparison of learning algorithms. The European METAL project [7] extended StatLog to cover more learning algorithms and more datasets, and investigated a number of other meta-features. Both projects sought to map meta-features to either a best performing algorithm or to a ranking of algorithms [4]. Another attempt to characterize meta-data through an analysis of meta-features is reported in [5]. In our case, we map datasets features to predictions on the benefits of using one learning algorithm in terms of different metrics, as execution time, accuracy or model understandability.

And third, from a planning perspective, we learn the values of some fluents defined in the domain by combining planning and execution. This is similar to the work reported in [14] where they automatically model the duration of actions execution as relational regression trees learned from observing plan executions. Those models are incorporated into the planning domain actions in the form of probabilistic conditional effects. In our case, we do not model probabilistic knowledge, but we learn multiple concepts in parallel. Also, they used relational models, while we learn propositional ones. The research reported in [13] explores the area of learning costs of actions from execution traces as well. They used predictive features of the environment to create situation-dependent costs for the arcs in the topological map used by a path planner to create routes for a robot. These costs are represented as learned regression trees.

7 Conclusions

This paper presents current work on the use of machine learning techniques to improve the KDD process of the PDM architecture. PDM uses automated planning techniques to help the data mining process. Previous work on PDM applied a learning technique to model the effect of selecting specific DM classification/regression actions to analyze a given dataset. However, those models were obtained from values resulting of applying KDD plans that incorporated not only the specific classification or regression techniques, but also a sequence of pre-processing techniques. In our current work, we are separating the execution of pre-processing actions from classification/regression actions, so that we can learn better predictive models of execution of KDD actions. We are also improving the features used for learning those models. In the future, we would like to provide the user with a mixed-initiative tool, so that the user can guide the KDD steps towards preferred techniques, or selecting or pruning KDD actions.

7.1 Acknowledgements

This work has been partially supported by the Spanish MICINN under projects TIN2008-06701-C03-03, TRA-2009-008, the regional projects CCG08-UC3M/TIC-4141 and the Automated User Knowledge Building (AUKB) project funded by Ericsson Research Spain.

References

[1] Robert S. Amant and Paul R. Cohen, 'Evaluation of a semi-autonomous assistant for exploratory data analysis', in *Proc. of the First Intl. Conf. on Autonomous Agents*, pp. 355–362. ACM Press, (1997).

[2] Abraham Bernstein, Foster Provost, and Shawndra Hill, 'Toward intelligent assistance for a data mining process: An ontology-based approach for cost-sensitive classification', *IEEE Transactions on Knowledge and Data Engineering*, **17**(4), (2005).

[3] Pavel Brazdil, Christophe Giraud-Carrier, Carlos Soares, and R. Vilalta, *Metalearning: Applications to Data Mining*, Cognitive Technologies, Springer, January 2009.

[4] Pavel Brazdil, Carlos Soares, and Joaquim P. Costa, 'Ranking learning algorithms: Using ibl and meta-learning on accuracy and time results', *Machine Learning*, **50**(3), 251–277, (March 2003). ISI, DBLP.

[5] Ciro Castiello, Giovanna Castellano, and Anna Maria Fanelli, 'Meta-data: Characterization of input features for meta-learning', in *MDAI*, pp. 457–468, (2005).

[6] Tomás De la Rosa, Angel García-Olaya, and Daniel Borrajo, 'Using cases utility for heuristic planning improvement', in *Case-Based Reasoning Research and Development: Proceedings of the 7th International Conference on Case-Based Reasoning*, pp. 137–148, Belfast, Northern Ireland, UK, (August 2007). Springer Verlag.

[7] Metal: A meta-learning assistant for providing user support in machine learning and data mining, 1998-2001.

[8] Susana Fernández, Fernando Fernández, Alexis Sánchez, Tomás de la Rosa, Javier Ortiz, Daniel Borrajo, and David Manzano, 'On compiling data mining tasks to pddl', in *Proceedings of International Competition on Knowledge Engineering for Planning and Scheduling, ICAPS'09*, Thessaloniki (Greece), (September 2009).

[9] M. Fox and D. Long, 'PDDL2.1: An extension to PDDL for expressing temporal planning domains', *Journal of Artificial Intelligence Research*, 61–124, (2003).

[10] Mark Hall Geoffrey Holmes and Eibe Prank, 'Generating rule sets from model trees', *Advanced Topics in Artificial Intelligence*, **1747**, 1–12, (1999).

[11] Malik Ghallab, Dana Nau, and Paolo Traverso, *Automated Planning - Theory and Practice*, Morgan Kaufmann, San Francisco, CA 94111, 2004.

[12] Alex Guazzelli, Michael Zeller, Wen-Ching Lin, and Graham Williams, 'PMML: An Open Standard for Sharing Models', *The R Journal*, **1**(1), 60–65, (May 2009).

[13] Karen Zita Haigh and Manuela M. Veloso, 'Learning situation-dependent costs: Improving planning from probabilistic robot execution', in *In Proceedings of the Second International Conference on Autonomous Agents*, pp. 231–238. AAAI Press, (1998).

[14] Jesús Lanchas, Sergio Jiménez, Fernando Fernández, and Daniel Borrajo, 'Learning action durations from executions', in *Proceedings of the ICAPS'07 Workshop on Planning and Learning*, Providence, Rhode Island (USA), (2007).

[15] Jun Won Lee and Christophe G. Giraud-Carrier, 'New insights into learning algorithms and datasets', in *ICMLA*, pp. 135–140, (2008).

[16] *Machine Learning, Neural and Statistical Classification*, eds., Donald Michie, D. J. Spiegelhalter, C. C. Taylor, and John Campbell, Ellis Horwood, Upper Saddle River, NJ, USA, 1994.

[17] Katharina Morik and Martin Scholz, 'The miningmart approach to knowledge discovery in databases', in *In Ning Zhong and Jiming Liu, editors, Intelligent Technologies for Information Analysis*, pp. 47–65. Springer, (2003).

[18] Vid Podpecăn, Nada Lavrač, Joost N. Kok, and Jeroen de Bruin, eds. *SoKD-09, "Third Generation Data Mining: Towards Service-oriented Knowledge Discovery", International Workshop on Third Generation Data Mining at ECML PKDD 2009*, Bled, Slovenia, September 2009.

[19] Saharon Rosset, Claudia Perlich, and Bianca Zadrozny, 'Ranking-based evaluation of regression models', *Knowledge and Information Systems*, **12**(3), 331–353, (2007).

[20] R. Vilalta, Christophe Giraud-Carrier, Pavel Brazdil, and Carlos Soares, 'Using meta-learning to support data mining', *International Journal of Computer Science and Applications*, **1**(1), 31–45, (2004).

[21] Ian H. Witten and Eibe Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd Edition, Morgan Kaufmann, 2005.

Supporting Users in KDD Processes Design: a Semantic Similarity Matching Approach

Claudia Diamantini and **Domenico Potena** and **Emanuele Storti** [1]

Abstract. Data Mining has reached a quite mature and sophisticated stage, with a plethora of techniques to deal with complex data analysis tasks. In contrast, the capability of users to fully exploit these techniques has not increased proportionately. For this reason the definition of methods and systems supporting users in Knowledge Discovery in Databases (KDD) activities is gaining increasing attention among researchers. The present work fits into this mainstream, proposing a methodology and the related system to support users in the composition of tools for forming valid and useful KDD processes. The basic pillar of the methodology is a similarity matching technique devised to recognize valid algorithmic sequences on the basis of their input/output pairs. Similarity is based on a semantic description of algorithms, their properties and interfaces, and is measured by a proper evaluation function. This allows to rank the candidate processes, so that users are provided with a criterion to choose the most suitable process with respect to their requests.

1 Introduction

Nowadays, Knowledge Discovery in Databases (KDD) is an open and dynamic market: whereas on the one hand the spreading of new technologies leverages the demand for advanced data analysis, on the other hand researchers continuously introduce new, typically more effective and more complex discovery techniques, algorithms and tools. In such a market, the socio-technological gap increases: data analysts are typically domain experts but naive KDD users that are not able to properly exploit state-of-the-art technologies. Although they may consult external KDD specialists, this is not always possible for small organizations with a limited budget. This scenario is shifting KDD research from delta-improvements of specific algorithms, to global KDD process design and management, supporting domain experts as well as KDD specialists in the choice of suitable algorithms for the given task, to compose these algorithms in a useful and valid way, and to execute the whole process.

In order to support the design of a KDD process, in this work we present a semantic similarity approach to algorithms matching. Given two algorithms, their match is based on the comparison between the output of the first and the input of the second, in order to determine whether they can be executed in sequence or not, i.e. whether their interfaces are compatible. Exploiting a semantically rich description of algorithms and reasoning capabilities, we are able to extend the definition of compatibility from exact match between input and output, to a *similarity matching criterion*. This criterion is based on subsumption relations and, unlike many previous works, parthood relations among a compound data structure and its subcomponents. Constraints are also introduced in order to guarantee semantic correctness of matches. We also introduce a similarity match evaluation function allowing to score the quality of a match. The matching criterion and the related cost function are exploited to define a semantic-based goal-driven procedure aimed at automatically generating *prototype KDD processes*, that is process models described as workflows of algorithms. Processes are also globally ranked so that users are provided with a criterion to choose the most suitable process with respect to their requests. An advantage of working at an algorithm level is that prototype KDD processes are general and reusable. Moreover, they can be considered as useful, valid and novel knowledge. On the contrary, since prototype KDD processes cannot be directly executed, each of them needs to be further elaborated by replacing each algorithm with a concrete software implementation. We conceived this approach to process composition in the framework of service-oriented KDD platforms, hence software implementations can be searched through available service registries.

In the next Section, we introduce semantic matching and the similarity evaluation function. The composition procedure is described in detail in Section 3, and in Section 4 the prototype of a system implementing the procedure is illustrated. Section 5 is devoted to discuss relevant related work in the Literature. Finally, Section 6 ends the paper.

2 Establishing Semantic Similarity

A KDD process is a workflow of algorithms, and is built for achieving a specific goal by means of various manipulations of a given dataset. Hence, our approach in process composition starts from the goal and goes backwards iteratively adding one or more algorithms to the process, until the head of the process is not able to directly elaborate the given dataset. The basic issue in composition is to define the algorithms matching criteria, that is to specify under which conditions two or more algorithms can be executed in sequence. These criteria have to ensure that the process is valid and semantically correct, i.e. that algorithms are syntactically interfaceable, and that the use of an algorithm's output as input to the next algorithm makes sense in the KDD domain. In some cases even if two algorithms are not syntactically interfaceable, knowledge about them and in particular about their I/Os allows integration after some data manipulations (e.g. by transforming a labeled dataset in an unlabeled one). In order to take into account the semantics of algorithms, we base our composition approach on semantic similarity matching criteria, which in turn are defined on a conceptualization of the KDD domain.

[1] Dipartimento di Ingegneria Informatica, Gestionale e dell'Automazione "M. Panti", Università Politecnica delle Marche, Ancona, Italy, email:{diamantini,potena,storti}@diiga.univpm.it

Table 1. Some KDDONTO classes.

Name	Description	Examples
Algorithm	algorithm for data analysis	SVM, RemoveMissingValues,PrincipalComponentAnalysis
Method	technique to extract knowledge	KernelMethod, RandomFill, FeatureExtraction
Phase	step in a KDD Process	Modeling, PreProcessing, FeatureExtraction
Task	Data Mining Task	Classification, Regression
Data	I/O Data	Dataset, LearningRate, Model
Model	type of I/O Data	Any induced classification model
Dataset	collection of data records	Iris dataset
DataFeature	characteristics of data	Numeric, Literal, MissingValues, BalancedDataset, NormalizedDataset

Table 2. Some KDDONTO relationships.

Name	Description	Example
uses(Algorithm, Method)	relation between an algorithm and the used method	uses(BACKPROP, NEURALNETWORK)
specifies_task(Method, Task)	relation between method and a task	specifies_task(NEURALNETWORK, CLASSIFICATION) specifies_task(NEURALNETWORK, REGRESSION)
specifies_phase(Task,Phase)	a task to a phase	specifies_phase(CLASSIFICATION, MODELING)
has_input(Algorithm∪Method∪Task, Data,Datafeature,strenght,is_parameter)	an input to an algorithm, method or task. DataFeature are preconditions on input Data. strenght defines whether the precondition is mandatory. is_parameter defines whether the input is a parameter	has_input(SOM, UnlabeledDataset, NO_MISS_VALUE,1,0) has_input(SOM, VectorQuantizer, FLOAT, 0.4, 0) has_input(SOM, LearningRate, null, null, 1) has_input(CLASSIFICATION, Data)
has_output(Algorithm∪Method∪Task, Data, DataFeature)	an output for an algorithm, method or task. DataFeature are postconditions resulting from data manipulation by the algorithm	has_output(SOM, VectorQuantizer, NO_LITERAL) has_output(CLASSIFICATION, Data)
is_a(Thing, Thing)	the generic subsumption relation between a class and its superclass	is_a(ClassificationAlgorithm, Algorithm) is_a(Model, Data), is_a(Dataset, Data), is_a(Parameter, Data)
part_of(Data, Data)	the relation between two data, such that the second contains the first as a subcomponent	part_of(Label, LabeledDataset) part_of(Neuron, MLP)
in_module/out_module(Algorithm, Algorithm)	explicit suggestion (best practice) in linking together two algorithms	in_module(LVQ, SOM) out_module(SOM, DRAWVORONOIREGIONS)
in_contrast(DataFeature, DataFeature)	disjoint properties of data	in_contrast(NUMERIC, LITERAL)

2.1 Conceptual Framework

The KDD domain has codified a set of rules to generate valid and useful knowledge using a number of existing data manipulation tools. These range from trivial constraints due to peculiar characteristics of a tool (e.g. remove missing values if the tool deals with vector data), to best practices as the CRISP-DM model. This core of knowledge has been formalized in the KDDONTO ontology [8], which describes algorithms, their interfaces, data and models, their properties and relationships, at an abstract level. Tables 1 and 2 summarize the basic concepts and relations of the ontology used in this work.

Note that, while classes and binary relationships have a direct counterpart in the OWL implementation of KDDONTO, n-ary relationships are transformed by the usual reification strategy. Moreover, in `has_input`, `has_output` relationships, *union* means that multiple classes can act as domain. This is implemented by using the owl:unionOf construct in the definition of the class. Hereafter, by the term *algorithm* we mean an instance of the homonymous class `Algorithm`, whereas terms input and output represent instances of the class `Data`; instances will be written in capitals. For further details about KDDONTO classes and relations and its OWL implementation we refer the interested reader to [8].

Our goal is to find, given the algorithm B, a set of algorithms $\{A_1,...,A_n\}$, whose output data can be used as input to B.

In order to give an illustrative example, let us consider the part of process shown in Figure 1. Suppose that an user wants to perform a classification task, and she can provide a dataset containing only float values with the exception of few missing values. At some point in such a process, a SOM algorithm has to be executed. SOM requires as input: an Unlabeled Dataset (`UDS`) with only float values and no missing values, an initial Vector Quantization (`VQ`) model and a Learning Rate (`LR`). Hence, we need algorithms that produce these data and whose use makes sense in KDD domain. In order to clean the dataset and remove missing values, a RemoveMissingValues (RMV) algorithm can be used. Such an algorithm takes as input a Labeled Dataset (`LDS`) and produces in output a `LDS` with `no_missing_values`. Therefore, the output of RMV and the input of SOM are not exactly the same datum (`LDS` vs. `UDS`), but they are compatible anyway: in fact, from their ontological description in KDDONTO we know that `LDS` is `composed_of` a `UDS` and a labeling function (`L`). Therefore, only the `UDS` subcomponent is given

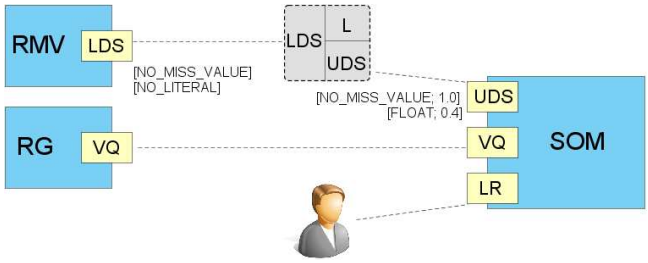


Figure 1. An example of similarity match: UDS is part of LDS, and $\texttt{VQ}_{RG}$ is equivalent to $\texttt{VQ}_{SOM}$. For each datum in square brackets pre- and post-conditions are shown. The number near to each precondition represents its strength.

in input to SOM, as represented by the dotted square in Figure 1. Moreover, even if RMV's output is compatible with SOM's input, we still need to check whether the former's postconditions are compatible with the latter's precondition. In detail, RMV returns a LDS with no_missing_values and only no_literal values, while SOM requires a UDS with two preconditions: the dataset must contain no_missing_values but only float values. The first condition is already satisfied. On the contrary, about the second condition, no_literal is more general than float; since such precondition is not mandatory (strenght less than 1), it can be relaxed; for these reasons the two algorithms can be linked together. Finally, a Random Generator algorithm (RG) is exploited to initialize the VQ, and the LR is manually set by the user.

In order to define a matching function that takes into account these kind of properties, we introduce a logic predicate and a score function. Both are based on KDDONTO: the former defines the matching criterion, whereas the latter evaluates the quality of matches.

2.2 Matching Criterion

Let in^i_B be the i^{th} input of the algorithm B, that is an element of class Data such that has_input(B, in^i_B). Similarly, $out^j_{A_k}$ is the j^{th} output of the algorithm A_k. A *similarity match* between a set of algorithms $\{A_1,...,A_n\}$ and an algorithm B is defined as:

$$\begin{aligned}&\textbf{match}_S(\{A_1,...,A_n\},B) \leftrightarrow \\ &\quad \forall\, in^i_B\ (is\_parameter(in^i_B) \vee \exists! A_k\ \exists! out^j_{A_k} : \\ &\quad valid(out^j_{A_k}, in^i_B)\ \wedge similar(out^j_{A_k}, in^i_B))\end{aligned} \tag{1}$$

The predicate $is\_parameter(in^i_B)$ returns the value is_parameter in the has_input relationship. If an input is a parameter, it is optional, but not mandatory, to find an algorithm which provides such a datum as output: as a matter of fact, a parameter is usually provided by the user. The predicate $valid(out^j_{A_k}, in^i_B)$ is satisfied if none of the postconditions of $out^j_{A_k}$ are in_contrast with any of the preconditions of in^i_B, that is:

$$\begin{aligned}&valid(out^j_{A_k}, in^i_B) \leftrightarrow \\ &\quad \nexists prec^h, postc^m : has\_input(B, in^i_B, prec^h) \wedge \\ &\quad has\_output(A_k, out^j_{A_k}, postc^m) \wedge \\ &\quad in\_contrast(prec^h, postc^m).\end{aligned}$$

The similarity predicate $similar(out^j_{A_k}, in^i_B)$ is satisfied:

- either if $out^j_{A_k} \equiv_o in^i_B$: they are conceptually equivalent, i.e., they refer to the same ontological concept;
- or if $out^j_{A_k} \sim_o in^i_B$: they are similar concepts, i.e. certain kinds of path between them exist in the ontology or can be inferred.

The ontological paths considered include is_a and part_of relations. Previous work about matchmaking commonly take into account only is_a relations, however generalization/specialization is not enough to exploit complex data structures which are common in KDD; for such a reason we consider similarity also at a *structural* level: a compound datum can be made of simpler data, according to the part_of relationship; see the discussion about UDS and LDS in the example of Figure 1.

Although equivalent and similar I/O pairs lead to valid matches, similarity should be weighted differently, because any path in the ontology implicitly introduce a data transformation in order to obtain an executable process. This and other issues are considered in the next subsection.

2.3 Cost of a Match

For each algorithm B, several sets of algorithms $\{A_1, ..., A_n\}$ satisfy the similarity match. In order to evaluate the quality of such matches, it is useful to assign them a numeric score such that the higher is the score, the less accurate is the match. In order to define such a score, hereafter we propose a cost function that depends on (1) the similarity among I/O data, (2) the relaxation of precondition constraints on input data of B, (3) best practices on linking A_k to B, and (4) algorithms' performances.

The *similarity between two I/O data* is determined by the number of is_a and part_of relations needed to link them in the ontology. In Figure 1, $\texttt{VQ}_{SOM}$ and $\texttt{VQ}_{RG}$ refer to the same ontological concept, so their degree of similarity is maximum; instead, $\texttt{UDS}_{SOM}$ and $\texttt{LDS}_{RMV}$ are similar in the sense that they are linked through a parthood relation. In general, given two data out_A and in_B, a *path* is the shortest sequence of is_a or part_of edges that are needed to move from the concept representing in_B to the concept related to out_A in the ontological graph. The similarity between out_A and in_B is measured by the *ontological distance* between them, which is defined as the weighted sum of the edges in the path:

$$D_o(out_A, in_B) = \sum_{i=1}^{|path|} \eta_i,$$

where η_i is the weight of the i^{th} edge, and its value depends on the type of relation. In particular, we set $\eta_{spec(ialization)} < \eta_{part(hood)} < \eta_{gen(eralization)}$. $D_o(out_A, in_B) = 0$ if the two data are conceptually equivalent. Unlike many previous works (e.g. [2, 4]), we weight differently a generalization and a specialization. To explain the reason of this asymmetry let us consider an algorithm B requesting a decision tree model (DT) as input. We could match B either with the algorithm A_1 returning a binary decision tree, which is a specialization of a DT, or with any other A_2 providing a classification model. While in the former case the matching is perfectly valid, the latter case implies the introduction of some operations that transform the generic classification model into a DT; of course this is not always possible, thus leading to non-executable processes. For such a reason $\eta_{spec} < \eta_{gen}$. Data manipulation implied by part_of relations are simpler to perform and always possible, hence $\eta_{part} < \eta_{gen}$.

The matching criterion entails that, for each pair (out_A, in_B), postconditions of out_A must not be in contrast with preconditions

of in_B; this allows to match data even if postconditions of out_A are not identical to preconditions of in_B (if preconditions are non-mandatory). *Relaxation of precondition constraints* has been performed in the example in order to link $\texttt{UDS}_{SOM}$ to $\texttt{LDS}_{RMV}$. We relaxed the constraint on the kind of data (float) requested by the SOM, and this implies that a dataset of type integer, as well as of type float, can be used as input to the SOM. Relaxing preconditions increases the cost of the match, because in such cases algorithm execution may lead to lower quality outcomes. In order to take into account the cost of relaxing preconditions of in_B, we define the function $\beta(out_A, in_B)$ as follows:

$$\beta(out_A, in_B) = \frac{1}{n_{in_B}} \sum_{h=1}^{n_{in_B}} \delta(out_A, p^h_{in_B}) \frac{S^2_{p^h_{in_B}}}{1 - S^2_{p^h_{in_B}}},$$

where n_{in_B} is the number of preconditions of in_B, $S_{p^h_{in_B}}$ is the strength of the precondition $p^h_{in_B}$, and δ is a function that takes the value 0 if no relaxation is performed, 1 otherwise. In practice δ is computed by evaluating the in_contrast relation among homogeneous DataFeature instances. In the example, NO_MISS_VALUE and FLOAT belongs to different subclasses, hence the two constraints do not affect each other. As concerns FLOAT and NO_LITERAL, an in_contrast property does not exist, hence the precondition can be relaxed. In this case $\delta = 1$. The cost of the preconditions relaxation depends on their strengths. In the example, since the strength of the FLOAT precondition of $\texttt{UDS}_{SOM}$ is 0.4, we have $\beta(\texttt{LDS}_{RMV}, \texttt{UDS}_{SOM}) = 0.1905$; and since $\texttt{VQ}_{RG}$ satisfies the precondition on $\texttt{VQ}_{SOM}$, $\beta(\texttt{VQ}_{RG}, \texttt{VQ}_{SOM}) = 0$. The choice of the specific non-linear function penalizes very much the choice of algorithms whose output data require the relaxation of near-mandatory preconditions (e.g., $S_{p^h_{in_B}} \geq 0.7$). If a mandatory precondition is relaxed β tends to $+\infty$.

The sum of D_o and β for each pair (out_{A_k}, in_B) is the cost of combining the interfaces of $\{A_1, ..., A_n\}$ and B. In order to define the cost of the match we take into account also functions that evaluate other characteristics of algorithms: $\alpha(A_k, B)$ evaluates *best practices on linking A_k and B*, and $\gamma(A_k)$ evaluates *performances* of A_k. Both of these functions act as corrective factors of the whole cost. In the former case, given two algorithms linked through the in/out_module relations, the cost is decreased because these relations state that a specific link between them was proved to be effective; in detail, if $out\_module(A_k, B)$ then $\alpha(A_k, B)$ is set to a value < 1, else it is equal to 1. In the latter case, performances can affect the cost of match, e.g. the higher the computational complexity, the higher the cost. $\gamma(A_k)$ is a function of A_k's performances. At present, performances are considered into KDDONTO but are not used for process composition, i.e. we set a constant value $\gamma(.) = 1$ for every algorithms.

Finally, given a match between a set of algorithms $\{A_1,...,A_n\}$ and B, we define the *match cost* function, with range $[0, +\infty)$, as follows:

$$C_M(\{A_1, ..., A_n\}, B) = \sum_{k=1}^{n} \alpha(A_k, B)\gamma(A_k) \cdot \frac{1}{m_k} \sum_{i=1}^{m_k} (D_o(out^j_{A_k}, in^i_B) + \beta(out^j_{A_k}, in^i_B)) \quad (2)$$

where m_k is the number of inputs of the algorithm B that are matched according to the criterion (1). Note that m_k is less than or equal to the number of inputs of B, because some parameters could be not considered in the match. The cost of match for the example is computed as follows:
$D_o(\texttt{LDS}_{RMV}, \texttt{UDS}_{SOM}) = \eta_{part} = 1$,
$D_o(\texttt{VQ}_{RG}, \texttt{VQ}_{SOM}) = 0$,
$\beta(\texttt{LDS}_{RMV}, \texttt{UDS}_{SOM}) = 0.1905$,
$\beta(\texttt{VQ}_{RG}, \texttt{VQ}_{SOM}) = 0$.

Supposing that no in/out_module among these algorithms exist, (i.e. $\alpha(., \texttt{SOM}) = 1$), the whole cost of match is:
$C_M(\{\texttt{RMV}, \texttt{RG}\}, \texttt{SOM}) = 1 \cdot (1 + 0.1905)_{RMV} = 1.1905$

3 Process Composition

Based on algorithms matching criteria, in this section we briefly describe a goal-driven procedure for composing KDD processes, which is formed of two main phases: (I) dataset and goal definition, (II) process building and ranking.

Dataset and goal definition. In our framework, a dataset is described by a set of characteristics that are instances of DataFeature class. The user goal is expressed as an instance of the Task class, leaving the user to move from complex domain-dependent business goal to a well-defined and domain-independent KDD task. The description of both dataset and goal allows to guide the composition procedure, bounding the number and type of algorithms that can be used at the beginning and at the end of each process. Moreover, in order to define a balance between procedure execution speed and composition accuracy, the user can provide some *process constraints*, e.g. maximum ontological distance for each match in a process (max_{Do}), maximum cost of a process (max_C), and maximum number of algorithms in a process (max_N).

Process building and ranking. Process building is an iterative phase, which starts from the given task and goes backwards iteratively adding one or more algorithms to a process according to the matching function defined in the previous section. The main steps in process building phase are described in Table 3.

A process $P_i = < \mathcal{V}_i, \mathcal{E}_i, C_i >$ is represented as a directed acyclic graph, where $\mathcal{V}_i$ is the set of nodes, namely algorithms, and $\mathcal{E}_i$ is the set of directed edges linking algorithms together. The number C_i represents the overall cost of the process P_i, which is computed as the sum of the cost of each match occurring in P_i plus the cost $\gamma(A_i)$, accounting the performance of the last algorithm in the process (i.e. the first algorithm the procedure has found).

At the beginning, algorithms A_i, which return as output a model x used for performing the given task T, and which uses classification methods are found; then, for each of them a candidate process P_i is created. If P_i does not violate process constraints (i.e. **process_constraints**(P_i) is true), it is added to the set $\mathcal{P}$ which contains all the valid processes that are going to be evaluated in the next step. At the beginning of each iteration, P_i is checked against the characteristics of the dataset at hand: if they are compatible, P_i is added to the set $\mathcal{F}$ of final processes. Note that a final process could be further expanded (e.g. we can add a feature selection algorithm for improving performances of a classification process); for this reason we do not immediately remove a final process from the set of valid ones. While there is a process P_i in $\mathcal{P}$, algorithms matchable with the one(s) at the head of P_i are extracted and used for forming a new candidate process. At last, P_i is deleted from $\mathcal{P}$ because its expansion has ended, and the procedure is iterated. As output of the com-

Table 3. The composition procedure.

Let $\mathcal{P}$ be the set of valid processes at each iteration, P_i be the i^{th} process in $\mathcal{P}$, described by the tern $<\mathcal{V}_i, \mathcal{E}_i, C_i>$ where $\mathcal{V}_i$ is the set of algorithms in P_i, $\mathcal{E}_i$ is the set of directed edges (A_p,A_q) which connect algorithm A_p to algorithm A_q, and C_i is the cost associated with the process. Let $\mathcal{F}$ be the final list of valid generated processes, T be the task and $\mathcal{D}$ be the set of dataset characteristics.

Let **match**$_D(\mathcal{D},P_i)$ be a predicate, which is true only if the set of inputs in_H of all algorithms at the head of P_i is such that $in_H \equiv_o$ DATASET and the preconditions of in_H are compatible with $\mathcal{D}$, or **is_parameter**(in_H).

Let **process_constraints**(P_i) be a predicate, which is false if P_i violates one of the process constraints.

$\mathcal{P} \leftarrow \varnothing$, $\mathcal{F} \leftarrow \varnothing$;
Find the set Γ=$\{A_i$: **has_output**$(A_i$,x) $\sqcap$ **output_for**(x,T) $\sqcap$ **uses**$(A_i$,m) $\sqcap$ **ClassificationMethod**(m)$\}$;
foreach $A_i \in \Gamma$ **do**
  Define P_i=$<A_i,\varnothing,\gamma(A_i)>$;
  if (**process_constraints**(P_i)) **then** $\mathcal{P} \leftarrow P_i$;

foreach $P_i \in \mathcal{P}$ **do**
  if (**match**$_D(\mathcal{D},P_i)$) **then** $\mathcal{F} \leftarrow P_i$;
  Define the set Δ=$\{B_k \in \mathcal{V}_i$: $\nexists$ (x, $B_k) \in \mathcal{E}_i\}$; */* B_k are the heads of P_i */*
  foreach $B_k \in \Delta$ **do**
    Find the set Φ=$\{\Phi_1,...,\Phi_m\}$, where Φ_j is the set of algorithms $\{A_1,...,A_{m_j}\}$
      such that **match**$_S(\Phi_j,B_k)$;
    foreach $\Phi_j \in \Phi$ **do**
      Define P'=$<\mathcal{V}_i \leftarrow \Phi_j$, $\mathcal{E}_i \leftarrow$$\{(A_1,B_k),...,(A_{m_j},B_k)\}$,$C_i + C_M(\Phi_j, B_k)>$;
      if (**process_constraints**(P')) **then** $\mathcal{P} \leftarrow P'$;
  $\mathcal{P}$=$\mathcal{P}$-$\{P_i\}$;

Sort $P_i \in \mathcal{F}$ by C_i.

position procedure, in $\mathcal{F}$ we have all the generated processes ranked according to their costs.

Besides to rank processes, we exploit costs of matches as a strategy for reducing the search space, and hence to control the complexity of the whole procedure. As a matter of fact, each time a new set of algorithms is added to a process, if the new cost is greater than max_C then the process is discarded. This is tantamount to pruning a whole subtree of possible processes.

The complete procedure contains an element that has not been introduced in Table 3 for the sake of simplicity. This element, called *precondition propagation* allows to solve a tricky problem related to the treatment of underspecified preconditions in the match function. To illustrate the problem, let us consider an algorithm B with a mandatory precondition *NO_LITERAL* on the input DATASET, and an algorithm A_1 with an output DATASET with no postconditions. In this case, a process matching A_1 and B should be discarded because the mandatory precondition is not satisfied. Let us now consider an algorithm A_2 with DATASET as output and postcondition NUMERIC: the sequence $A_2 \rightarrow A_1$ definitely produces a numeric dataset, hence the process $A_2 \rightarrow A_1 \rightarrow B$ becomes valid. In order to take into account this kind of processes, preconditions that are not explicitly violated are stored and checked successively. In particular, if an A_k exists such that $similar(out^j_{A_k}, in^i_B)$ and in turn A_k has an input that is conceptually equivalent to $out^j_{A_k}$, then A_k inherits the precondition. In this way, the process is not discarded and the decision on its validity is postponed to the next iteration of the composition procedure.

4 KDDComposer tool

The procedure for composing KDD processes has been implemented in the software prototype *KDDComposer*, with the aim to test its effectiveness. KDDComposer is a Java application, deployed as an Eclipse plug-in, which allows to manage the whole composition procedure by a GUI. For managing the interaction between the application and KDDONTO we use Jena[2] as a framework for querying the ontology through SPARQL language [1], which is a W3C Recommendation; Pellet[3] is used as reasoner for inferring non-explicit facts. Figure 2 shows the experiment creation tab. The tool dynamically retrieve all the possible tasks and dataset characteristic from the ontology, allowing the user to choose the most suitable for her purposes. Furthermore, in the left-bottom box process constraints can be set, in order to define a balance between number of generated processes and execution speed. In the example reported in Figure, the user set the *classification* task, on a *labeled dataset* with the following characteristics: data of type *float*, *normalized* and with *missing values*; the maximum number of algorithms in a process is limited to 5; the maximum ontological distance for each match is 3, and the maximum cost of a process is set to 10. Results are available in Figure 3: each execution of the procedure, stored into a database, is shown in the top-left box. Choosing a specific experiment, in the middle-left box, all generated processes are listed and ranked according to their process cost. By selecting a process, its graph is visualized in the center of the window: each node is an algorithm, an edge represents a match between two algorithms and can include one or more

[2] http://jena.sourceforge.net/
[3] http://clarkparsia.com/pellet

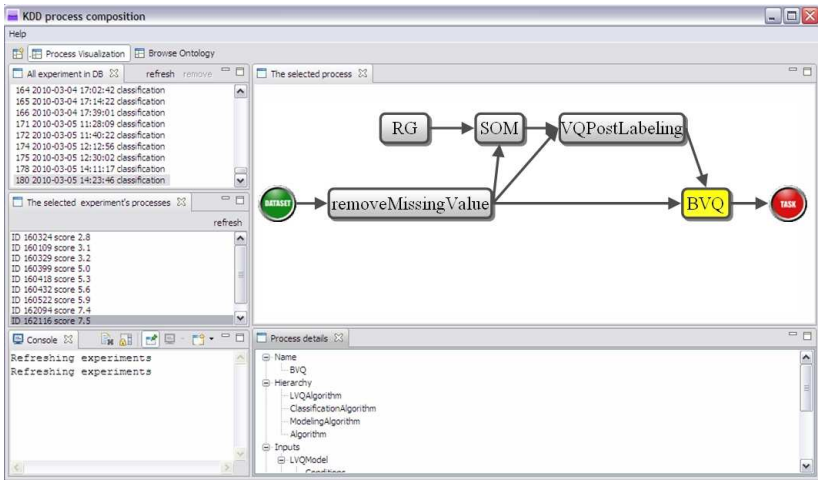


Figure 3. KDDComposer: a generated process.

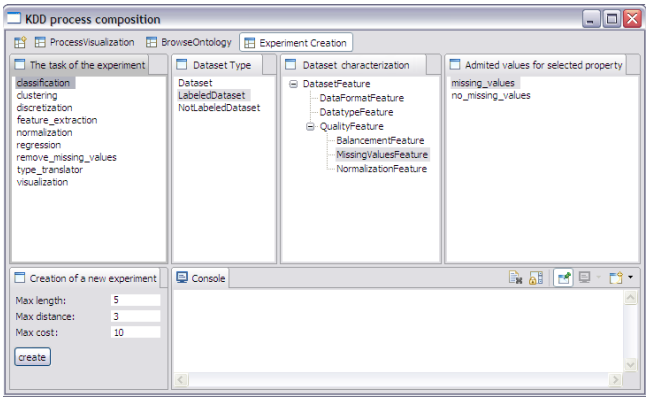


Figure 2. KDDComposer: setting up the procedure.

input/output pairs. Further information on algorithms (name, I/O interface, method) and on edges (type of data, pre/postconditions) are available in the bottom box. The resulting process shown in Figure 3 is made of 5 algorithms, linked through both exact and similarity matches. In more detail, the classification task is performed by the BVQ algorithm, which produces in output a Labeled Vector Quantization (`LVQ`) model. BVQ requires as input a Labeled Dataset (`LDS`) without missing values and a `LVQ` for initialization purpose. While `LDS` (without missing values) can be obtained from a RemoveMissingValues algorithm, the `LVQ` for initialization is obtained from the VQPostLabeling, which takes as input a `LDS` and a Vector Quantization `VQ` model, thus defining a label for each vector quantizer. At this point, the `LDS` is already available, and can be taken from the RemoveMissingValues algorithm, while the `VQ` is obtained by the SOM algorithm. In turn, this last requires as input an Unlabeled Dataset (`UDS`) (without missing values) and a `VQ` for initialization. Since `UDS` is `part_of` `LDS`, there is a similarity match between RemoveMissingValues and SOM. Finally, the `VQ` to initialize SOM is provided by a RandomGenerator (`RG`). In Table 4 details about the interface of each tool are provided, together with the mandatory preconditions and postconditions of each data. We'd like to note that all the preconditions (in this case about data type and data quality) are satisfied: since the dataset has *missing values*, the RemoveMissingValues algorithm is introduced in order to satisfy SOM's and BVQ's precondition, while no algorithm is introduced to transform data types, because the dataset contains only *float* values and all the algorithms involved in the process accept them.

Algorithm	input	output
RG	-	VQ
RMV	LDS[1]	LDS[2]
SOM	UDS[3], VQ	VQ
VQPostLabeling	VQ, LDS	LVQ
BVQ	LDS[4], LVQ	LVQ

Pre/Postconditions:
1 : missing_values
2 : no_missing_values, no_literal
3 : no_missing_values, float
4 : no_missing_values, no_literal

Table 4. Details about the process in Figure 3.

The KDDComposer prototype is available at the KDDVM project site [13]. At present our implementation of KDDONTO ontology describes 15 algorithms of preprocessing, modeling and postprocessing phases, in particular for Feature Extraction, Classification, Clustering, Evaluation and Interpretation tasks.

5 Related Work

In last years researchers in Data Mining and KDD fields have shown more and more interest in techniques for giving support in the design of knowledge discovery processes [3, 10, 16, 19, 21]. An early work of this kind is [10], where authors suggests a framework for guiding users in breaking up a complex KDD task into a sequence of manageable subtasks, which are then mapped to appropriate Data Mining techniques. Such an approach was exploited in [19], where the user is supported in iteratively refining a KDD skeleton process, until executable techniques are available to solve low-level tasks. To this end, algorithms and data are modeled into an object oriented schema. In

[16] a system is described focusing on setting-up and reusing chains of preprocessing algorithms, which are represented in a relational meta-model.

Although these works help users in choosing the most suitable tools for each KDD phase, no automatic composition procedure is defined.

Recent work have dealt with this issue [3, 21, 14] in a way similar to our proposal. In detail, in [3] authors define a simple ontology (actually not much more than a taxonomy) of KDD algorithms, that is exploited to design a KDD process facing with cost-sensitive classification problems. The procedure generates all possible valid processes but only in a linear, non-splitting form. [21] introduces a KDD ontology of concrete implementations of algorithms, datasets and models, which is exploited to guide a forward state-space search for the design of a KDD workflow. Such a procedure returns the best process, and the selection criteria is the minimum number of steps. In [14] is proposed a procedure for composing KDD processes using a Hierachical Task Network planning (HTN), which is guided by a Data Mining ontology. In such a way the process is generated by decomposing an initial task into applicable operators and sub-tasks.

Differing from our proposal, both in [3] and in [21], ontologies are not rich enough to be extensively used for deducing hidden relations and they do not consider similarity matches. Unlike [21], we consider several information to define an evaluation function to rank processes; our backwards approach generates both linear and non-linear processes (unlike [3]), without imposing (like in HTN decomposition) limitations on the usage of algorithms in different parts of the process: for instance, even if a SOM algorithm is typically used to perform a clustering task, our procedure can select it to preprocess a dataset, in order to initializate a classification algorithm (e.g., BVQ). Moreover, our approach, being based on a conceptualization of algorithms and not on their implementation, is aimed to achieve a higher level of generality, by producing abstract and reusable KDD prototype processes. They can be seen as suggested sequences of the conceptual steps that should be performed to achieve the requested goal, starting with a specific dataset.

Some other ontologies have been also defined, focusing only on tools and algorithms for Data Mining, which is one of the phases of the wider and more complex KDD field [11, 7]. The first ontology of this kind is *DAMON* (DAta Mining ONtology) [6], that is built for simplifying the development of distributed KDD applications on the Grid, offering to domain experts a taxonomy for discovering tasks, methods and software suitable for a given goal. In [20], the ontology is exploited for selecting algorithms on the basis of the specific application domain they are used for. Finally, *OntoDM* [18] is a general purpose top-level ontology aimed to describe the whole Data Mining domain.

Some related works can be found in the Web Service field. Most of them consider only the subsumption relation for matching services [2, 5, 12, 17]. Very few proposals deal with only exact match, while most of them consider also the subsumption relation for matching services, e.g. [2, 5, 12, 17, 15]. To the best of our knowledge the only previous approach considering also parthood in matchmaking is [4], which describes a procedure for Web Service discovery using semantic matchmaking. To this end, authors exploit relations of the lexical ontology Wordnet, including the *meronymy* relation (i.e. part-of), but matchmaking is discussed only within a Web Services discovery framework, without dealing with composition issues. As concerns match cost functions, [2, 4] weight generalization/specialization relations in the same way, while different weights are proposed in [5, 12, 17]. These works do not take into account further contribution to the final score such as preconditions, linkable modules and performances. A preliminary version of the matching criterion and the composition procedure is proposed in [9]. With respect to such a work, the present proposal extends the matching criterion by introducing the management of input parameters, and improves the definition of the composition procedure by fixing some issues like precondition propagation and by exploiting the cost function in order to reduce the complexity of the procedure and to rank processes. The cost function is defined for the first time in this work.

6 Conclusion

The main contribution of this work is the introduction of a semantic matching function for the automatic composition of algorithms forming valid prototype KDD processes. The proposed procedure is based on the exploitation of KDDONTO, that formalizes knowledge about KDD algorithms. Our approach has manifold advantages. Firstly, the resulting processes are valid and semantically correct. Secondly, unlike other work in the Literature, we are able to generate implicit, interesting and non-trivial processes where algorithms share similar interfaces. These processes can be themselves considered as useful, valid and valuable knowledge, both for novice and expert users. Furthermore, the introduction of a cost function allows to rank processes according to both ontological and non-ontological measures, and it is also exploited for reducing the search space, and hence for controlling the complexity of the composition procedure. A prototype system implementing the whole procedure is also presented.

As a future work, we are currently working on enhancing the descriptive capabilities of KDDONTO ontology, by adding further information about algorithms and their interfaces. In particular, our analysis is focusing on describing, from an ontological point of view, statistical characteristics of datasets (e.g., distribution, skew, curtosis). Such information can be exploited to improve the suggestion about which algorithm is likely to perform better with the dataset at hand. Since the output of our procedure are abstract KDD processes, it is not possible to directly execute them, and a further step is needed, namely the translation into a concrete process of tools. Once obtained concrete processes, we plan to design comprehensive experimentations, in order to evaluate the effectiveness of the composition procedure and of the cost functions.

REFERENCES

[1] . SPARQL Query Language for RDF, W3C Recommendation. http://www.w3.org/TR/rdf-sparql-query/, 2008.

[2] Akkiraju, R., Srivastava, B., Ivan, A., Goodwin, R. and Syeda-Mahmood, T., 'SEMAPLAN: Combining Planning with Semantic Matching to Achieve Web Service Composition', in *Proc. of the IEEE International Conference on Web Services (ICWS 2006)*, pp. 37–44, Chicago, USA, (2006).

[3] Bernstein, A., Provost, F. and Hill, S., 'Toward Intelligent Assistance for a Data Mining Process: An Ontology Based Approach for Cost-Sensitive Classification', *IEEE Transactions on Knowledge and Data Engineering*, **17**(4), 503–518, (2005).

[4] Bianchini, D., De Antonellis, V. and Melchiori, M., 'Flexible semantic-based service matchmaking and discovery', *World Wide Web*, **11**(2), 227–251, (2008).

[5] Budak Arpinar, I., Aleman-Meza, B., Zhang, R. and Maduko, A., 'Ontology-driven Web services composition platform', in *Proc. of IEEE International Conference on e-Commerce Technology*, pp. 146–152, San Diego, CA, USA, (2004).

[6] Cannataro, M. and Comito, C., 'A data mining ontology for grid programming', in *Proc. 1st Int. Workshop on Semantics in Peer-to-Peer and Grid Computing, in conjunction with WWW2003*, pp. 113–134, Budapest, Hungary, (2003).

[7] CRISP-DM site. . http://www.crisp-dm.org.
[8] Diamantini, C., Potena, D. and Storti, E., 'KDDONTO: an Ontology for Discovery and Composition of KDD Algorithms', in *Proc. of the ECML/PKDD09 Workshop on Third Generation Data Mining: Towards Service-oriented Knowledge Discovery*, pp. 13–24, Bled, Slovenia, (Sep 7-11 2009).
[9] Diamantini C., Potena D., and Storti E., 'Ontology-driven KDD Process Composition', in *Proc. of the 8th International Symposium on Intelligent Data Analysis*, ed., Adams, N. et al., volume 5772 of *LNCS*, 285–296, Springer, Lyon, France, (Aug 31 - Sep 2 2009).
[10] Engels, E., 'Planning tasks for knowledge discovery in databases; performing task-oriented user-guidance', in *Proceedings of the 2nd International Conference on Knowledge Discovery in Databases (KDD'96)*, Portland, Oregon, (August 1996).
[11] Fayyad, U.M., Piatetsky-Shapiro, G. and Smyth, P., *From data mining to knowledge discovery: an overview*, 1–34, American Association for Artificial Intelligence, Menlo Park, CA, USA, 1996.
[12] Fuji, K. and Suda, T., 'Dynamic Service Composition using Semantic Information', in *Proc. of 2nd International Conference on Service Oriented Computing*, pp. 39–48, New York City, NY, USA, (2004).
[13] KDDVM project site. http://boole.diiga.univpm.it.
[14] Kiets, J., Serban, F., Bernstein, A. and Fisher, S., 'Towards Cooperative Planning of Data Mining Workflows', in *Proc. of the ECML/PKDD09 Workshop on Third Generation Data Mining: Towards Service-oriented Knowledge Discovery*, pp. 1–12, Bled, Slovenia, (Sep 7-11 2009).
[15] Lemmens, R. and Arenas, H., 'Semantic Matchmaking in Geo Service Chains: Reasoning with a Location Ontology', *International Workshop on Database and Expert Systems Applications*, **0**, 797–802, (2004).
[16] Morik, K. and Scholz, M., 'The MiningMart Approach to Knowledge Discovery in Databases', in *Intelligent Technologies for Information Analysis*, ed., Zhong, N. and Liu, J., 47–65, Springer, (2004).
[17] Ni, Q., 'Service Composition in Ontology enabled Service Oriented Architecture for Pervasive Computing', in *Workshop on Ubiquitous Computing and e-Research*, Edinburgh, UK, (2005).
[18] Panov, P., Džeroski, S. and Soldatova, L., 'OntoDM: An Ontology of Data Mining', in *Data Mining Workshops, International Conference on*, pp. 752–760, Los Alamitos, CA, USA, (2008). IEEE Computer Society.
[19] Wirth, R., Shearer, C., Grimmer, U., Reinartz, T., Schlösser, J.J., Breitner, C., Engels, R. and Lindner, G., 'Towards Process-Oriented Tool Support for Knowledge Discovery in Databases', in *PKDD '97: Proceedings of the First European Symposium on Principles of Data Mining and Knowledge Discovery*, pp. 243–253, London, UK, (1997). Springer-Verlag.
[20] Yu-hua, L., Zheng-ding, L., Xiao-lin, S., Kun-mei, W. and Rui-xuan, L., 'Data mining ontology development for high user usability', *Wuhan University Journal of Natural Sciences*, **11**(1), 51–56, (2006).
[21] Žáková, M., Křemen, P., Železný F. and Lavrač, N., 'Using Ontological Reasoning and Planning for Data Mining Workflow Composition', in *SoKD: ECML/PKDD 2008 workshop on Third Generation Data Mining: Towards Service-oriented Knowledge Discovery*, Antwerp, Belgium, (2008).

Monte-Carlo Tree Search: From Playing Go to Feature Selection

Michele Sebag

LRI (Laboratoire de Recherche en Informatique), Universite Paris Sud, France

(Invited Talk)

Abstract

This talk will show that some key learning tasks (focussing on Feature Selection; but the same applies to Active Learning as well) can be formalized as Reinforcement Learning (RL) problems: ultimately, the goal is to find a sequence of decisions (a policy) leading to a minimal generalization error. While finding an optimal policy is utterly intractable, as could have been expected, this optimal policy can be approximated within a one-player game framework.

The proposed approach is built on Monte-Carlo Tree Search, and specifically the Upper Confidence Tree (UCT) proposed by Kocsis and Szepesvari (2006). Reusing the same UCT skeletton as used in MoGo, the world champion Computer-Go program devised by Gelly, Teytaud et al., the FUSE algorithm achieves Feature Selection, yielding state-of-art results on the NIPS Feature Selection Challenge (2003).

Among the specific extensions needed to adapt UCT to the Feature Selection game are:

- designing a reward (fast and unbiased estimate of the generalization error)
- dealing with a many armed bandit problem (the number of arms is the number of features) and controlling the exploration vs exploitation tradeoff;
- dealing with an unknown finite horizon (the target number of relevant features).

References

1. Romaric Gaudel and Michele Sebag. Feature Selection as a One-Player Game. *In Proc. of ICML 2010*
2. Koesis, L. and Szepesvári, C. Bandit based Monte-Carlo planning. *In ECML'06, pp. 282–293. Springer Verlag, 2006.*
3. Rolet, P., Sebag, M., and Teytaud, O. Boosting Active Learning to optimality: a tractable Monte-Carlo, Billiard-based algorithm. *In ECML/PKDD'09, pp. 302–317. Springer Verlag, 2006.*

Collaborative Meta-Learning

Joaquin Vanschoren [1] and **Larisa Soldatova** [2]

Abstract. This paper proposes an ontology and a markup language to describe machine learning experiments in a standardized fashion and support a *collaborative* approach to the analysis of learning algorithms. Experiments can then be shared with the community and augmented with large amounts of experiments by other researchers as well as all measurable properties of algorithms and datasets. By maximally reusing what has been learned before, by many researchers, this approach enables machine learning researchers to perform in-depth meta-learning studies with minimal effort, e.g., to investigate and explain algorithm performance on different types of data and under different parameter settings. As can been learned from recent developments in other sciences, such a free exchange and reuse of experiments requires a clear representation. We therefore focus on the design of ontologies and formal representation languages to express and share machine learning meta-data with the world.

1 Motivation

1.1 Experimentation in machine learning

Research in machine learning is inherently empirical. Researchers, as well as practitioners, seek a deeper understanding of learning algorithm performance by performing large numbers of learning experiments. Whether the goal is to develop better learning algorithms or to select useful approaches to analyze new sources of data, collecting the right meta-data and correctly interpreting it is crucial for a thorough understanding of learning processes.

However, running those experiments tends to be quite laborious. In the case of evaluating a new algorithm, pictured in Figure 1, one needs to search for datasets, preprocessing algorithms, (rival) learning algorithm implementations and scripts for algorithm performance estimation (e.g. cross-validation). Next, one needs to set up a wide range of experiments: datasets need to be preprocessed and algorithm parameters need to be varied, each of which requires much expertise, and experiment designs [17] need to be employed to thoroughly test the influence of different experimental variables. Finally, after all experiments have run, one needs to properly organize all the collected results in order to interpret them correctly. This easily amounts to a large range of experiments representing days, if not weeks of work, while only averaged results will ever be published. Any other researcher willing to verify some results or test a certain hypothesis will have to start again from scratch, repeating the same experiments instead of simply reusing them.

1.2 Generalizability and Interpretability

Moreover, in order to ensure that results are generally valid, the empirical evaluation also needs to be equally general, meaning that it must cover many different conditions. These include various parameter settings and various kinds of datasets, e.g. differing in size, skewness, noisiness or with or without being preprocessed with basic techniques such as feature selection. Unfortunately, because of the amount of work involved in empirical evaluation, many studies will not explore these conditions thoroughly, limiting themselves to testing algorithms (often only with default parameter settings) on a selection of benchmark datasets. It has long been recognized that such studies are in fact only 'case studies' [1], and should be interpreted with caution.

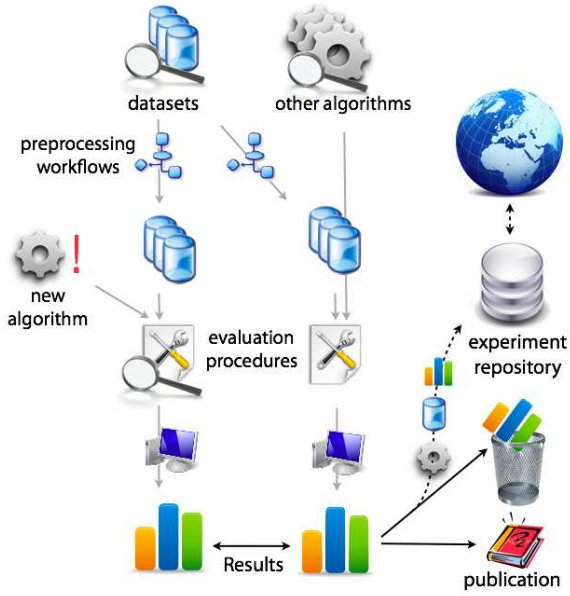


Figure 1. A typical experiment workflow in machine learning research.

Indeed, sometimes, overly general conclusions can be drawn. First, in time series analysis research, many studies were shown to be biased toward the datasets being used, leading to contradictory results [16]. Second, it has been shown that the relative performance of learning algorithms can depend heavily on the amount of sampled training data (their learning curves cross) [21, 28], and is also easily dominated by the effect of parameter optimization and feature selection [13]. Since only few studies thoroughly vary data sample sizes, parameter settings or feature sets, it is not clear how generally valid their results are.

As such, there are good reasons to thoroughly explore different conditions, or at least to clearly state under which conditions certain conclusions may or may not hold. Otherwise, it is very hard for other researchers to correctly interpret the results, thus possibly creating a false sense of progress [10]:

[1] K.U. Leuven, Belgium, email: joaquin.vanschoren@cs.kuleuven.be
[2] Aberystwyth University, UK, email: lss@aber.ac.uk

...no method will be universally superior to other methods: relative superiority will depend on the type of data used in the comparisons, the particular data sets used, the performance criterion and a host of other factors. [...] an apparent superiority in classification accuracy, obtained in laboratory conditions, may not translate to a superiority in real-world conditions...

1.3 Large-scale studies

Several comprehensive empirical studies exist that try, as well as possible, to cover a wide range of conditions. The StatLog [18] and MetaL [5] projects and, more recently, some large empirical studies, for instance Ali and Smith [2] and Caruana and Niculescu [6], aim to extract very general conclusions from large and very general experiment setups. Still, as new algorithms, preprocessing methods, learning tasks, and evaluation metrics are introduced at a constant rate, it is impossible for a single study to cover this continuously expanding space of learning approaches. Moreover, the meta-data generated by these and thousands of other machine learning studies is usually collected and stored differently and therefore hard to share and reuse.

2 A collaborative approach

In this paper, we advocate a much more dynamic, *collaborative* approach to experimentation, in which experiments can be freely shared, linked together, augmented with measurable properties of algorithms and datasets, and immediately reused by researchers all over the world. Any researcher creating empirical meta-data should thus be able to easily share it with others and in turn reuse any prior results of interest. Indeed, by reusing prior results we can avoid unnecessary repetition and speed up scientific research, and by bringing the results of many studies together, we can obtain an increasingly detailed picture of learning algorithm behavior. In turn, this facilitates large-scale, very generalizable machine learning studies which are prohibitively expensive to start from scratch.

2.1 e-Sciences

The use of such public experiment repositories is common practice in many other scientific disciplines, such as micro-array repositories in bio-informatics [25] and virtual observatories in astrophysics [26]. These so-called e-Sciences aim to share as much empirical data as possible online, creating an "open scientific culture where as much information as possible is moved out of people's heads and labs, onto the network and into tools that can help us structure and filter the information" [19].

Ironically, while machine learning and data mining have been very successful in speeding up scientific progress in these fields by discovering useful patterns in a myriad of collected experiment results, machine learning experiments themselves are currently not being documented and organized well enough to engender the same automatic discovery of insightful patterns that may speed up the design of new algorithms or the selection of the best algorithms to analyze new collections of data.

2.2 An infrastructure for experiment exchange

Similar to developments in these e-Sciences, we need to clearly formalize machine learning techniques and investigations, and set up

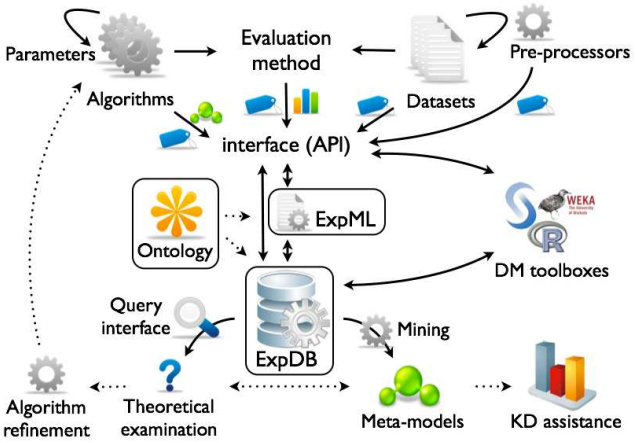


Figure 2. An infrastructure for experiment exchange.

online infrastructures for experiment exchange. The latter typically consist of more or less the same three components, shown in the boxed elements of Figure 2:

A formal representation language. To enable a free exchange of experiment data, a standard and formal representation language needs to be agreed upon. Such a language may also contain guidelines about the information necessary to ensure reproducibility.

Ontologies. Defining a coherent and unambiguous description language is not straightforward. It requires a careful analysis of all the concepts used within a domain and their relationships. The result can be formally represented in *ontologies* [7]: machine manipulable domain models in which each concept is clearly described. They establish a common vocabulary for describing experiments, so that experiments by other researchers can be clearly interpreted, even by software agents.

A searchable repository. To reuse experiment data, we need to locate it first. Experiment repositories therefore still need to organize all data to make it easily retrievable. Querying tools or query languages can be used to facilitate access.

To realize this system, we build upon *experiment databases* (ExpDBs) [3, 28, 27]: databases designed to collect the details of these experiments, and to intelligently organize them in online repositories to enable fast and thorough analysis of a myriad of collected results. While the design and use of experiment databases in machine learning has been amply discussed before, this paper will focus on the design of the remaining two components needed to extend and use them collaboratively. We will first introduce Exposé, a proposed ontology for machine learning experimentation, and next, we will illustrate how we can translate this ontology into a formal, XML-based representation language called ExpML.

To make the sharing of experiments as easy as possible, experiment descriptions should be generated automatically: a programming interface (API), shown in the top center of Figure 2 is provided that allows to build uniform representations of algorithm implementations, datasets and entire experiments. This interface can be included in data mining tools so that experiments can be shared (or downloaded) at the click of a button. Measurable properties of algorithms and datasets (shown as labels in Figure 2) can be added as well. Next, the experiments are automatically exported to a common format (ExpML) and organized in experiment databases. Such databases can be setup for personal use, to organize all of one's experiments, or to build lab-wide or world-wide repositories.

The arrows emanating from the ExpDB at the bottom of Figure 2 shows different ways to tap into the shared meta-data:

Querying allows a researcher to formulate questions about the stored experiments, and immediately get all results of interest. Several query interfaces have been developed, both online and in a downloadable tool, available on `http://expdb.cs.kuleuven.be`. They include a graphical query composer helping the user to quickly select experiment details or impose constraints, and offer various visualizations of the returned results.

Mining. A second use is to automatically look for patterns in algorithm performance by mining the stored meta-data. The insights provided by such *meta-models* can then be used to design better algorithms or to select and apply them in knowledge discovery applications [5].

Integration. Data mining toolboxes could also interface with ExpDBs directly. All experiments performed with the toolbox could be automatically exported to ExpDBs, and previous experiments, run before by a different user of that toolbox, can be downloaded to avoid repetition and speed up experimentation.

In the remainder of this paper, we will first outline Exposé, our proposed ontology, in Section 3, and the resulting XML-based language, ExpML, in Section 4. Finally, we provide some illustrations of possible meta-learning studies in Section 6. Section 7 concludes.

3 The Exposé ontology

Sharing machine learning experiments with each other starts with speaking the same language. Moreover, if we want to automatically organize thousands of experiments generated by various researchers all over the world, this language should be interpretable by machines. Designing a coherent and unambiguous formal language is not straightforward, especially since experimentation in machine learning is a very involved process including various statistical techniques and many different setups. Indeed, a very fine-grained description will be needed if we wish to answer questions about detailed aspects of the involved learning algorithms and datasets.

In this section, we briefly describe an ontology [7, 12], called Exposé, in which the concepts used within machine learning experiments and their relationships are carefully described and expressed in a formal domain model. It provides a common, unambiguous core vocabulary for researchers wishing to describe their experiments, and it explicates the inherent structure of machine learning experiments. Ontologies are a logical choice for the principled design of community-based experiment databases, since they are built to evolve: they can be modified, extended and refined by many different researchers to cover ever more types of experiments, tasks and algorithms. We can thus edit or extend our domain model on a conceptual level, and 'translate' these changes into updated markup languages and database models, so that they also evolve with the field.

In a way, we start small. We will focus on supervised classification and our experiments are limited to algorithm evaluations on static, propositional datasets. Still, this already covers a decent amount of contemporary experimentation and includes many concepts common to other subfields. It is important to note that this is a straw-man proposal that is intended to instigate discussion and attract wider involvement from the data mining community. It has been influenced and adapted by many people, mostly through close collaboration with the authors of other data mining ontologies.

Exposé is described in the OWL-DL ontology language [12]. It can be downloaded from the experiment database website (`http://expdb.cs.kuleuven.be`), and explored and edited using any OWL-DL editor, e.g. the Protégé editor (v4)[3].

In all, the ontology currently defines over 850 classes (concepts), most of which describe algorithms, dataset and algorithm characteristics, experiment design methods and evaluation functions.

3.1 Ontology design

In designing Exposé, we paid close attention to existing guidelines for ontology design [15]:

Top-level ontologies. It is considered good practice to start from generally accepted and unambiguously described classes (concepts) and properties (relationships) [20]. We started from the Basic Formal Ontology (BFO)[4] covering top-level scientific classes and the OBO Relational Ontology (RO)[5] offering a predefined set of relationships.

Ontology reuse. If possible, (parts of) other ontologies should be reused to build on the knowledge (and the consensus) expressed in those ontologies. When designing Exposé, we reused general machine learning related classes from the OntoDM ontology [20] (a general, high-level ontology for data mining with the aim of providing a unified framework for data mining research), experimentation-related classes from the EXPO ontology [24] (a top-level ontology for scientific experiments containing classes for hypotheses, (un)controlled variables, experiment designs and experiment equipment), and classes related to internal algorithm mechanisms from the DMOP ontology [11]. In fact, Exposé bridges the gap between the very specific classes of DMOP and the very general ones of OntoDM, thus providing an important contribution to the harmonization of various data mining ontologies. Any future extensions of any of these other ontologies can directly be used to update Exposé, and vice-versa.

Design patterns. Ontology design patterns[6] are reusable, successful solutions to recurrent modeling problems. For instance, we mentioned that a learning algorithm can act as an individual learner in one experiment, and as a base-learner for an ensemble learner in the next. This is a case of an agent-role pattern, in which an agent (algorithm) only plays a certain role in a process in some occasions, but not always. A predefined relationship, 'realizes', is used to indicate that an individual is able to fulfill a certain role. We have used such patterns as often as we could.

Quality criteria. General criteria include *clarity* (descriptions of the classes should make the meaning of each concept clear), *coherence* or *consistency* (there should be no logical contradictions), *extensibility* (future uses should be anticipated) and *minimal commitment* (only support the intended knowledge sharing activities). These criteria are rather qualitative, and were only evaluated through discussions with other researchers.

Many classes and properties were extracted from earlier working versions of our experiment database [3], and thus proved practically useful to organize experiment information. Vice versa, many limitations of those earlier versions were solved through Exposé's much more principled domain model.

[3] http://protege.stanford.edu/
[4] http://www.ifomis.org/bfo
[5] http://www.obofoundry.org/ro/
[6] http://ontologydesignpatterns.org

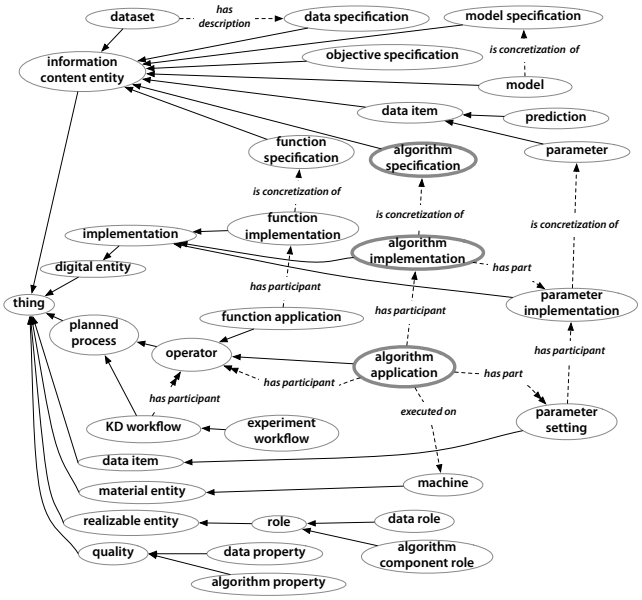


Figure 3. An overview of the top-level classes in the Exposé ontology.

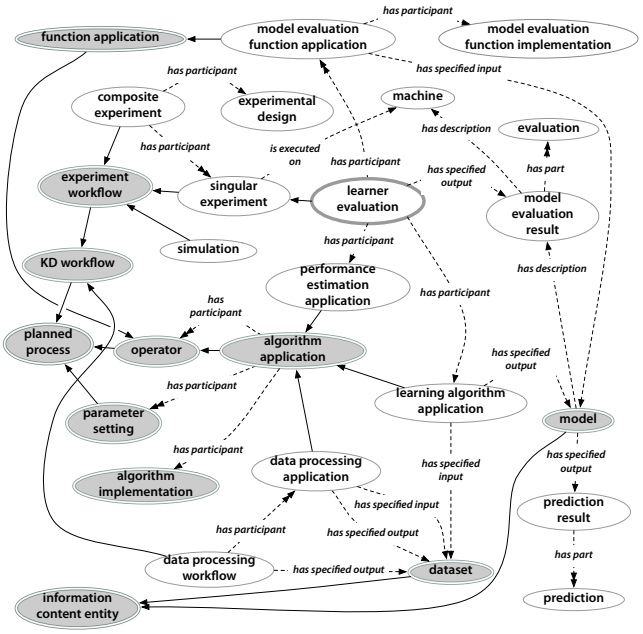


Figure 4. Experiments in the Exposé ontology.

3.2 Top-level View

Figure 3 shows the most important top-level classes and properties, many of which are inherited from the OntoDM ontology [20], which in turn reuses classes from OBI[7] (i.e., planned process) and IAO[8] (i.e. information content entity). The full arrows symbolize an 'is-a' relationship, meaning that the first class is a subclass of the second, and the dashed arrows symbolize other common relationships. Double arrows indicate one-to-many relationships, for instance, an *algorithm application* can have many *parameter settings*.

The three most important categories of classes are *information content entity*, which covers datasets, models and abstract specifications of objects (e.g. algorithms), *implementation*, and *planned process*, a sequence of actions meant to achieve a certain goal. When describing experiments, this distinction is very important. For instance, the class 'C4.5' can mean the abstract algorithm, a specific implementation or an execution of that algorithm with specific parameter settings, and we want to distinguish between all three.

As such, ambiguous classes such as 'learning algorithm' are broken up according to different interpretations (indicated by bold ellipses in Figure 3): an abstract *algorithm specification* (e.g. in pseudo-code), a concrete *algorithm implementation*, code in a certain programming language with a version number, and a specific *algorithm application*, a deterministic function with fixed parameter settings, run on a specific *machine* with an actual input (a *dataset*) and output (a *model*), also see Figure 4. The same distinction is used for other algorithms (for data preprocessing, evaluation or model refinement), mathematical functions (e.g. the kernel used in an SVM), and parameters, which can have different names in different implementations and different value settings in different applications. Algorithm and function applications are *operators* in a workflow (see below), and can even be participants of another algorithm application (e.g., a kernel or a base-learner), i.e. they can be part of the inner workflow of an algorithm.

Finally, there are also *qualities*, properties of a specific dataset or algorithm (see Figures 6 and 7), and *roles* indicating that an element assumes a (temporary) role in another process: an algorithm act as a base-learner in an ensemble, a function can act as a distance function in a learning algorithm, and a dataset can be a training set in one experiment and a test set in the next.

[7] http://obi-ontology.org
[8] http://code.google.com/p/information-artifact-ontology

3.3 Experiments

Figure 4 shows the ontological description of experiments, with the top-level classes from Figure 3 drawn in filled double ellipses. Experiments are defined as *workflows*. The general nature of workflows allows the description of many kinds of experiments. Some (composite) experiments can also consist of many smaller (singular) experiments, and can use a particular *experiment design* [17] to investigate the effects of various *experimental variables*, e.g. parameter settings.

We will now focus on a particular kind of experiment: a *learner evaluation* (indicated by a bold ellipse). This type of experiment applies a specific learning algorithm (with fixed parameters) on a specific input dataset and evaluates the produced model by applying one or several model evaluation functions, e.g. predictive accuracy. In predictive tasks, a performance estimation technique, e.g. 10-fold cross-validation, is applied to generate training- and test sets, evaluate the resulting models and aggregate the results. After it is executed on a specific *machine*, it will output a *model evaluation result* containing the outcomes of all evaluations and, in the case of predictive algorithms, the (probabilistic) predictions made by the models. Models are also generated by applying the learning algorithm on the entire dataset.

Finally, more often than not, the dataset will have to be preprocessed first. Using workflows, we can define how various *data processing applications* preprocess the data before it is passed on to the learning algorithm. Figure 5 illustrates such a workflow. The top of the figure shows that it consists of participants (operators), which in turn have inputs and outputs (shown in ovals): datasets, models and model evaluation results. Workflows themselves also have inputs and outputs, and we can define specialized sub-workflows. A *data processing workflow* is a sequence of data processing steps. The center of Figure 5 shows one with three preprocessors. A *learner evaluation*

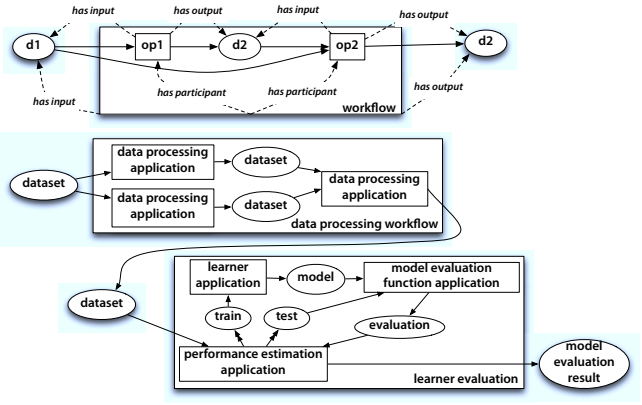


Figure 5. Experiment workflows.

workflow takes a dataset as input and applies performance estimation techniques (e.g. 10-fold cross-validation) and model evaluation functions (e.g. the area under the ROC curve) to evaluate a specific learner application.

This constitutes a clear definition of how the learning algorithm is evaluated, making sure it can be interpreted unambiguously. Of course, there are other types of learner evaluations, both finer ones, e.g. a singular train-test experiment, and more complex ones, e.g. doing an internal model selection to find the optimal parameter settings. Exposé also defines 7 evaluation techniques and around 50 different evaluation functions, which can in turn have different implementations.

3.4 Datasets

Figure 6 shows the most important classes used to describe datasets.

Specification. The *data specification* (in the top part of Figure 6) describes the structure of a dataset. Some subclasses are graphs, sequences and sets of instances. The latter can have instances of various types, for instance tuples, in which case it can have a number of *data features* and *data instances*. For other types of data (e.g. relational data) this specification will have to be extended. Finally, a dataset has descriptions, including the dataset name, version, download url and a textual description of its origin, which are needed to make the dataset easily retrievable.

Roles. A specific dataset can play different *roles* in different experiments (top of Figure 6). For instance, it can be a training set in one evaluation and a test set in the next.

Data properties. As said before, we wish to link all empirical results to theoretical metadata, called *properties*, about the underlying datasets to perform meta-learning studies. These *data properties* are shown in the bottom half of Figure 6, and may concern individual instances, individual features or the entire dataset. We define both *feature properties* such as feature skewness or mutual information with the target feature, as well as general *dataset properties* such as the number of attributes and landmarkers [22].

3.5 Algorithms

One last aspect of the ontology we wish to highlight is the description of algorithms. To allow the exploration of algorithm performance under different configurations and parameter settings, and include all aspects that influence their performance in queries, we need a detailed vocabulary to describe them.

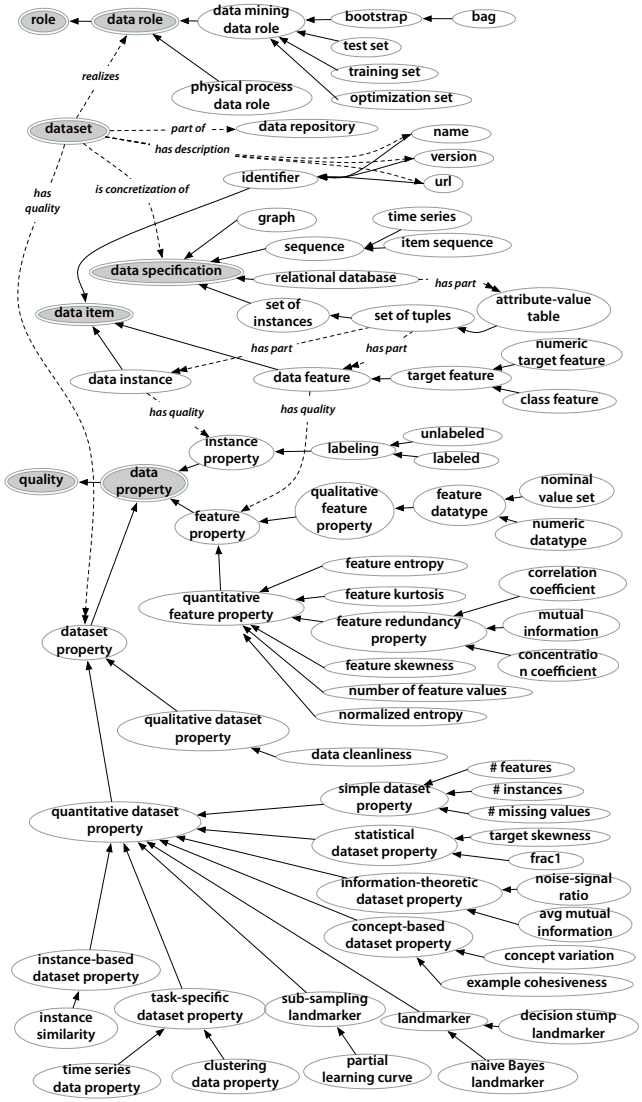


Figure 6. Datasets in the Exposé ontology.

Algorithm implementations. Figure 7 shows how algorithms and their configurations are expressed in our ontology. As mentioned in Section 3.2, algorithms can have various *implementations* and *applications* with specific *parameter settings*. The latter attach a value to a *parameter implementation*, which depending on the algorithm in which they are implemented, can have different names and different default values. Algorithm implementations are described with all information needed to retrieve and use them, such as their name, version, url, and the library they belong to (if any). Moreover, they can have *qualities*, e.g. their susceptibility to noise.

Algorithm composition. Some algorithms also use other algorithms or mathematical functions, which can often be selected (or plugged in) by the user. These include base-learners in ensemble learners, distance functions in clustering and nearest neighbor algorithms and kernels in kernel-based learning algorithms. Some algorithm implementations also use internal data processing algorithms, e.g. to remove missing values. In Exposé, any operator can be a participant of an algorithm application, combined in internal workflows with in- and outputs. Depending on the algorithm, operators can ful-

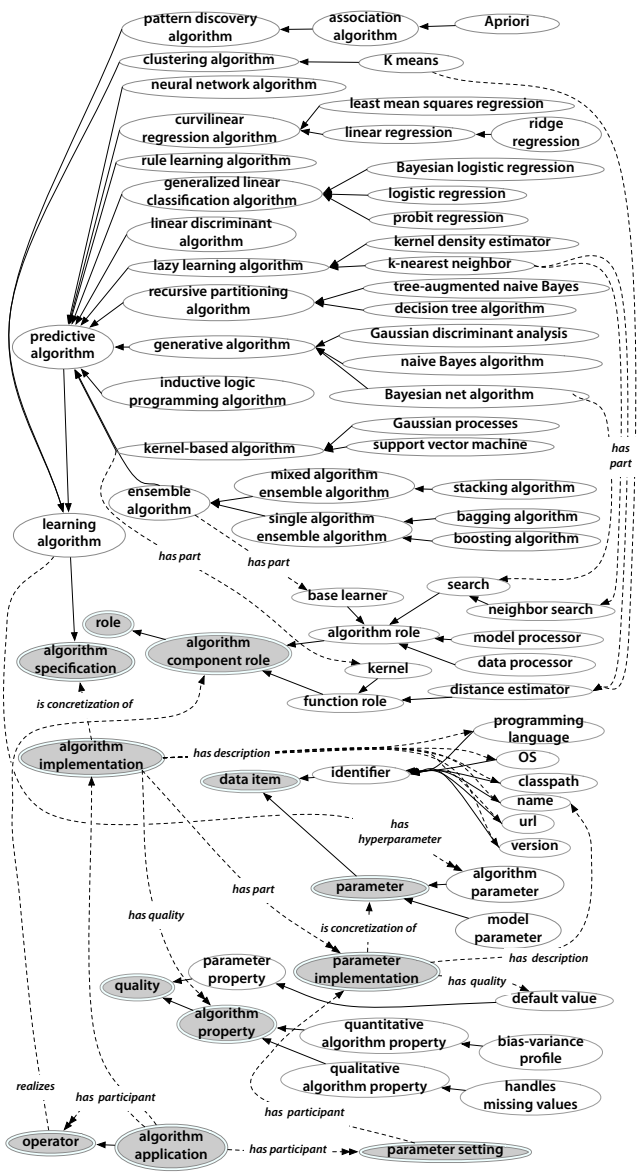


Figure 7. Algorithms and their configurations in the Exposé ontology.

fill (realize) certain predefined *roles* (center of Figure 7).

The top half of Figure 7 shows the classes of algorithms that are currently defined in the ontology, and which internal operators they have. For instance, a support vector machine (SVM) is always assumed to have a kernel, and the choice of kernel and its implementation will be fixed on application.

Algorithm mechanisms. Finally, to understand the performance differences between different types of algorithms, we need to look at the internal learning mechanisms on which they are built. These include the kind of models that are built (e.g. decision trees), how these models are optimized (e.g. the heuristic used, such as information gain) and the decision boundaries that are generated (e.g. axis-parallel, piecewise linear ones in the case of non-oblique decision trees). These classes, which extend the algorithm definitions through specific relationships (e.g. *has model structure*), are defined in the DMOP ontology [11], so they won't be repeated here.

4 The ExpML language

Using the Exposé ontology as our core vocabulary, we can define a formal markup language for describing experiments. This entails a translation of ontological classes and properties to XML elements and syntax. This translation process is especially useful because it allows ontological extensions (e.g. to new machine learning tasks) to be translated into updated ExpML definitions.

4.1 Operators and processes

First, we need to define which aspects of machine learning experiments we wish to share. We can divide these in two groups:

- Definitions of new experiment elements, such as new algorithms, datasets, evaluation functions and kernels. These correspond to algorithm or function *implementations* and *datasets* in our ontology.
- The experiment setups and results, corresponding to *planned processes*: *experiment workflows*, *data processing workflows*, and their in- and outputs, such as *model evaluation results*.

Because ontological relationships are more expressive than XML syntax, different relationships between these classes need to be translated quite differently. Table 1 provides a short overview of these relationships and their XML equivalent. Figures 8-10 illustrate this process, showing a real experiment (experiment 445080 in our experiment database) expressed in ExpML. We first describe a new algorithm implementation, and then we perform an experiment in which a dataset is preprocessed with a feature selection technique, and subsequently used to evaluate the added algorithm implementation.

4.2 New Operators

We start by describing a new algorithm implementation, i.e. WEKA's implementation of the bagging algorithm. Figure 8 shows the ExpML description, together with the corresponding Exposé classes above. First, we take the core class, *learning algorithm implementation*, and convert it into an XML element: *learner implementation*. Next, we express all its properties, as inherited from its parent, *algorithm implementation*.

4.3 Experiment Workflows

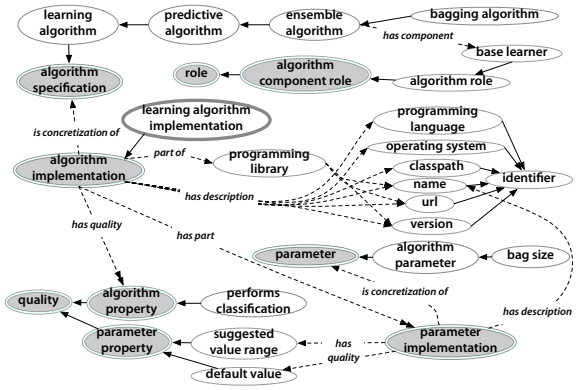
Figure 9 shows, from bottom to top, a simplified experiment workflow, the expression of the first half of this workflow in ExpML and the corresponding part of the ontology. To express workflows in ExpML, we assign an id to each operator and in- or output. For each operator, we state its inputs in an `input data` attribute, and for each output, we state the operator that generated it in an `output of` attribute. As shown in the ExpML code, a dataset `d1` is used as the input of workflow `op1` and data processing operator `op2`. The resulting dataset `d2` is referenced as both the output of operator `op1` and workflow `op2`.

The data processing workflow (Figure 9) contains a single participant: a data processing application, i.e. a feature selection algorithm. The ontology shows that this algorithm, in turn, requires an input, a dataset, and has a participant: a specific implementation. Since it is also an algorithm application, it can have multiple *parameter settings* and internal *operators*.

In this case, there are two operators: a feature subset evaluation function and a search algorithm, each *realizing* a certain *algorithm component role*. The first is realized by a *function application*,

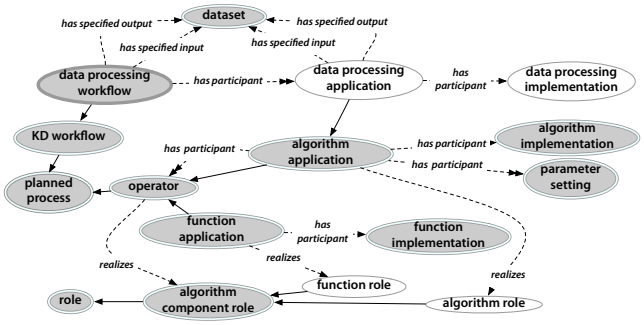
Table 1. Translating ontological properties to XML syntax.

Ontological property	XML syntax	example
has-part, pas-participant with role attribute if defined	target: subelement of source `parameter_impl` (Figure 8)	
has-description	(required) attribute	`name` attribute (Figure 8)
has-quality	subelement called `property`	`property` (Figure 8)
is-concretization-of	`implementation_of` attribute	`implementation_of` attribute (Figure 8)
part-of	specific attributes	`libname` and `libversion` (Figure 8)
has-specified-input	input given id, referenced in `input_data` attribute	`input_data='d1'` (Figure 9)
has-specified-output	source given id, referenced in `output_of` attribute	`output_of='d1'` (Figure 9)




```
<expml>
 <learner_impl name="weka.Bagging" version="1.31" os="any"
    url="http://..." language="java" libname="weka"
    implementation_of="bagging_algorithm"
    classpath="weka.classifiers.meta.Bagging">
  <parameter_impl name="P" implementation_of="bag_size">
    <property name="default_value">100</property>
    <property name="suggested_value_range">
        20,40,60,70,80,90,100</property>
  </parameter_impl>
  <parameter_impl name="I" implementation_of="
      number_of_iterations">
    <property name="default_value">10</property>
    <property name="suggested_value_range">
        10,20,40,80,160</property>
  </parameter_impl>
  <parameter_impl name="S" implementation_of="random_seed"
      >
    <property name="default_value">1</property>
  </parameter_impl>
  <learner_impl role="base-learner"/>
  <property name="performs_classification">true</property>
  <property name="handles_missing_values">true</property>
  ...
 </learner_impl>
```

Figure 8. An algorithm implementation, in Exposé and ExpML.




```
<dataset id="d1" name="kr-vs-kp" url="http://...">
 <target_feature name="class" index="-1" />
</dataset>
<data_processing_workflow id="op1" input_data="d1">
 <data_processing_appl id="op2" input_data="d1">
  <data_processing_impl name="AttributeSelection" version=
      "1.7" libname="weka"/>
  <function_appl role="feature_subset_evaluator">
   <function_impl name="CfsSubsetEval" version="1.26" />
  </function_appl>
  <search_algorithm_appl role="search_algorithm">
   <search_algorithm_impl name="BestFirst" version="1.2"/>
    <parameter_setting name="D" value="1"/>
    <parameter_setting name="N" value="5"/>
  </search_algorithm_appl>
 </data_processing_appl>
</data_processing_workflow>
<dataset id="d2" name="kr-vs-kp-AttrSelection-
    CfsSubsetEval-BestFirst-D1-N5" url="http://..."
    output_of="op1,op2">
 <target_feature name="class" index="-1"/>
</dataset>
```

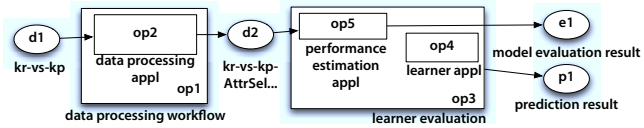


Figure 9. Data processing workflow in Exposé and ExpML

hence the inclusion of a `function appl` element with that role in the XML code. In turn, it has also a participant: a `function implementation`.

The second is a `search algorithm appl` that also has some parameter settings. Note that we make a small exception here: normally, a `parameter impl` subelement should be included in the `parameter setting`. Still, as an algorithm will only use its own parameter implementations, we chose for a more readable representation, in which simply the name is added as an attribute.

4.4 Experiment Evaluation and Results

As shown in the second half of the workflow at the bottom of Figure 9, the generated dataset serves as the input for a learner evaluation, which will in turn produce a model evaluation result and a prediction result.

The ExpML description is shown in Figure 10, and the corresponding ontological classes in Figure 4. It consists of a learning algorithm application complete with parameter settings and a base learner application with its own parameter settings. It also has a performance estimation technique (10-fold cross-validation) and a list of evaluation functions to assess the produced models, each pointing to their precise implementations. This level of detail is important to assess whether the results of this evaluation can be confidently reused. Indeed, there are many pitfalls associated with the statistical evaluation of algorithms [23, 8, 9]. By stating exactly which techniques were used, we can query for appropriate results. For instance, when comparing algorithms with cross-validation, it is important that the

```
<learner_evaluation id="op3" input_data="d2" series="exp1"
     experiment_id="445080">
 <learner_appl id="op4">
  <learner_impl name="weka.Bagging" version="1.31.2.2"
       libname="weka"/>
  <parameter_setting name="P" value="100"/>
  <parameter_setting name="I" value="10"/>
  <parameter_setting name="S" value="1"/>
  <learner_appl role="base-learner">
    <learner_impl name="weka.REPTree" version="1.19.2.2"
         libname="weka"/>
    <parameter_setting name="M" value="2"/>
    <parameter_setting name="V" value="0.0010"/>
    <parameter_setting name="N" value="3"/>
    <parameter_setting name="S" value="1"/>
    <parameter_setting name="L" value="-1"/>
  </learner_appl>
 </learner_appl>
 <performance_estimation_appl id="op5" input_data="d2">
  <performance_estimation_impl name="weka.
       crossValidateModel" version="1.53" libname="weka"/>
  <parameter_setting name="numfolds" value="10"/>
  <parameter_setting name="random" value="1"/>
 </performance_estimation_appl>
 <model_evaluation_function_appl name="predictive_accuracy
      ">
  <model_evaluation_function_impl name="weka.pctCorrect"
       version="1.53"  libname="weka"/>
 </model_evaluation_function_appl>
 ...
</learner_evaluation>
<model_evaluation_result  id="e1" output_of="op3,op5">
 <machine>vic_ni-09-10</machine>
 <evaluation name="build_cputime" value="2.25"/>
 <evaluation name="build_memory" value="36922896"/
 <evaluation name="mean_absolute_error" value="0.017"/>
 <evaluation name="root_mean_squared_error" value="0.085"
      />
 <evaluation name="predictive_accuracy" value="0.991"/>
 <evaluation name="confusion_matrix" value="[[won,nowin
      ],[1660,19],[9,1508]]"/>
 <evaluation name="precision_array" value="[[won,nowin
      ],[0.988,0.994]]"/>
 ...
</model_evaluation_result>
<prediction_result id="p1" output_of="op3">
  <prediction instance="0000" value="won">
    <probability outcome="won" value="0.985"/>
    <probability outcome="nowin" value="0.015"/>
  </prediction>
  ...
  <prediction instance="3195" value="nowin">
    <probability outcome="won" value="0.010"/>
    <probability outcome="nowin" value="0.990"/>
  </prediction>
</prediction_result>
```

Figure 10. An experiment workflow (setup and results) in ExpML.

same folds are used for all algorithms.

The output of the experiment is shown next, consisting of all evaluation results (also stating the machine used in order to interpret cpu time) and all predictions, including the probabilities for each class. Although omitted for brevity, evaluation error margins are stored as well. Storing predictions is especially useful if we want to apply new evaluation metrics afterwards without rerunning all prior experiments. We do not provide a format to describe models, as there already exist such formats (e.g. PMML).

5 Organizing Machine Learning Information

With thousands of ExpML files detailing performed experiments, the final step is to organize all this information in searchable databases so that it can be retrieved, rearranged, and reused in further studies. Although it is outside of the scope of this paper, we use the same ontological domain model to define a relational database model, so that future refinements of the domain model can be translated more easily to refinements of the database. This leads to a very fine-grained database model, warranting detailed queries about many different aspects of machine learning experiments. Currently, the database stores about 650,000 experiments on 54 classification algorithms, 87 datasets and 45 evaluation metrics. Simple data processing workflows, including feature selection and learning curve analyses are also used. All details can be found on the ExpDB website, at `http://expdb.cs.kuleuven.be`.

6 Meta-learning studies

We now illustrate the use of such databases for meta-learning investigations, increasingly making use of the available meta-level descriptions of algorithms and datasets. In each case, we use a single database query to generate the results. While we won't show the queries here, they can again be found on the ExpDB website. Most of these illustrations were presented before in Vanschoren et al. [28], and additional examples can be found there as well.

6.1 Ranking Algorithms

When selecting interesting learning approaches, or when testing the effectiveness of a new algorithm, one is often interested in a ranking of the available methods. To investigate whether some algorithms consistently rank high over various problems, we can query for their average rank (using each algorithm's optimal observed performance) over a large number of datasets, using optimized parameter settings. Figure 11 shows such a ranking over all UCI datasets.[9] To check which algorithms perform significantly different over many datasets, we used the Friedman test [8]. The right axis shows the average rank divided by the *critical difference*, meaning that two algorithms perform significantly different if the average ranks of two algorithms differ by at least that value (one unit).[10]

This immediately shows that indeed, some algorithms rank higher on average than others on the UCI datasets. Bagged naive Bayes trees seem to come in first overall, but the difference is not significant compared to that of SVMs with a polynomial kernel (although this SVM seems to outperform C4.5). Also note that bagging and boosting PART and NBTrees seem to yield big performance boosts, while boosting random trees proves particularly ineffective.

6.2 Tuning Parameters

By querying for the values of the parameter settings in each experiment, we can investigate their effects on several kinds of data. For instance, Figure 12 shows the effect of the kernel width (gamma) of the RBF kernel on a number of different datasets with the same default accuracy (10%). Note that on some datasets, increasing the gamma value will steeply increase performance, until it reaches a maximum and then slowly declines, while on other datasets, performance immediately starts decreasing up to a point, after which it quickly drops to default accuracy. Querying for the number of attributes in each dataset (shown in brackets), suggests that only datasets with few attributes benefit from large gamma values. Further investigation showed that the used SVM implementation easily starts overfitting when both gamma and the number of attributes are high [28]. This is an example of a query suggesting ways to improve an algorithm.

[9] We selected 18 algorithms to limit the amount of statistical error generated by using 'only' 87 datasets.

[10] The critical difference was calculated using the Nemenyi test with p=0.1, 18 algorithms and 87 datasets.

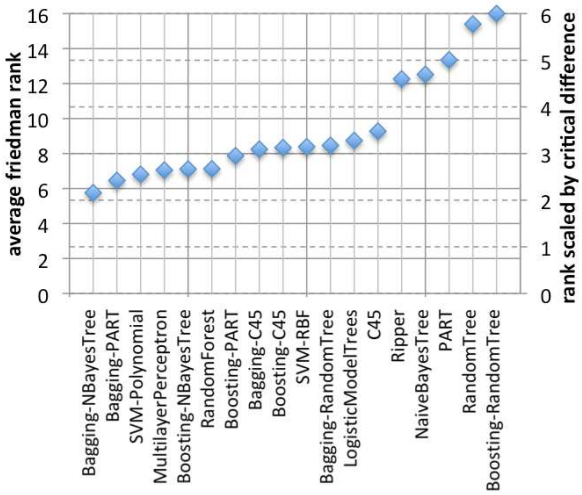


Figure 11. Average rank, specific algorithm setups.

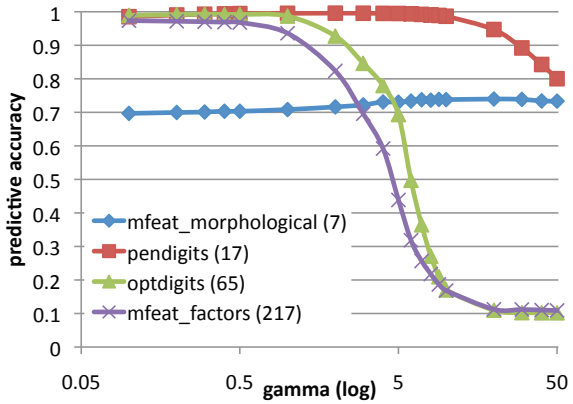


Figure 12. Effect of the RBF kernel width on different datasets.

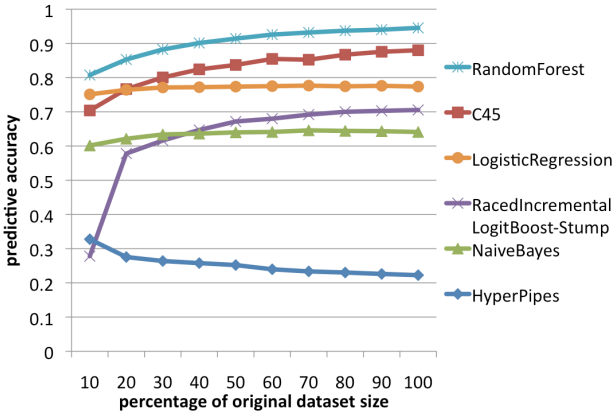


Figure 13. Learning curves on the Letter-dataset.

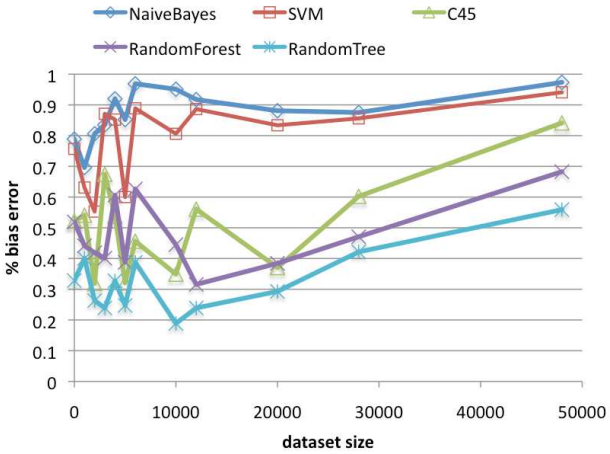


Figure 14. The average percentage of bias-related error in algorithms as a function of dataset size.

6.3 Applying Preprocessors

In many cases, data needs to be preprocessed before a certain learning algorithm can be used effectively. Since the database stores the entire experiment workflow, we can analyze the effect of preprocessing techniques on the performance of learning algorithms. For instance, to investigate how much data a certain algorithm needs to perform optimally, we can query for the results on downsampled versions of the dataset, yielding a learning curve for each algorithm, as shown in Figure 13. Note that some learning curves cross, indicating that the ranking of those algorithms depends on the size of the sample. While logistic regression is initially stronger than C45, the latter keeps on improving when given more data. However, to investigate the effect of preprocessing methods, the database model will need to be extended further.

6.4 Generalizing over Algorithms

We may also want to investigate which general properties of an algorithm might be desirable when approaching a certain task. One very interesting property of an algorithm is its bias-variance profile [14]. Since the database contains a large number of bias-variance decomposition experiments, we can give a realistic numerical assessment of how capable each algorithm is in reducing bias and variance error. The (maximum) percentage of bias-related error, compared to the total error, is stored in the database as an algorithm property.

We can also query for the effect of the dataset size on the dominance of the bias error component. Averaging the bias-variance results over datasets of similar size for each algorithm produces the result shown in Figure 14. It shows that bias error is of varying significance on small datasets, but steadily increases in importance on larger datasets, for all algorithms. For large datasets, it is thus advisable to choose a low-bias algorithm. This corroborates a previous study [4], but now on a much larger collection of datasets.

6.5 Mining for Patterns

Finally, instead of studying different dataset properties independently, we could also use data mining techniques to find patterns in the behavior of algorithm on various kinds of data. When querying for the default performance of OneR and J48 (WEKA's C4.5 implementation) on all UCI datasets, and plotting them against each other, we obtain Figure 15. It shows that on a large number of datasets, their performance is indeed about the same. Still, J48 works much better on many other datasets.

To automatically learn under which conditions J48 clearly outperforms OneR, we queried for the characteristics of each dataset, and discretized the data into three class values: "draw", "win_J48" (>4%

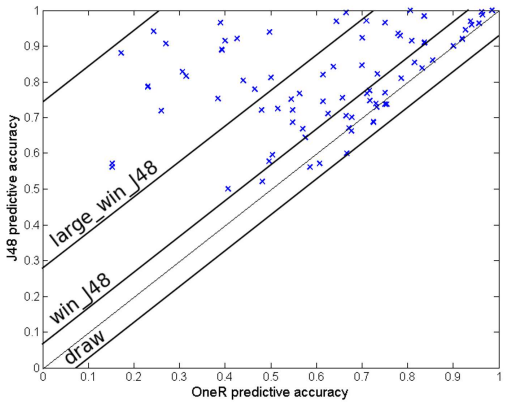


Figure 15. Performance comparison of J48 and OneR on all UCI datasets.

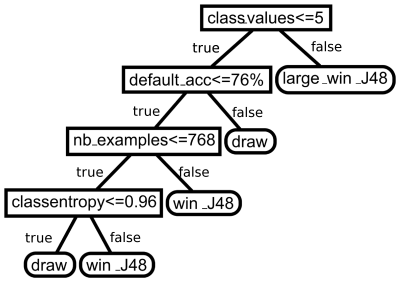


Figure 16. Meta-decision tree showing when J48 is better than OneR

gain), and "large_win_J48" (>20% gain). The tree returned by J48 on this meta-dataset is shown in Figure 16, showing that a high number of class values often leads to a large win of J48 over 1R.

Many more interesting meta-models could be discovered by simply querying the database and mining the results.

7 Conclusions

In this paper, we take an important step towards an infrastructure for collaborative experimentation in machine learning, in which experiments from many researchers can be automatically shared, organized, and analyzed in depth simply by querying experiment repositories. First, we describe the most important aspects of Exposé, a proposed ontology for machine learning experimentation. Next, we illustrate how we can translate this ontology into a formal XML-based language, ExpML, to describe experiments. Finally, we use an experiment database, also modeled after Exposé and filled with large amounts of classification experiments, to perform various in-depth meta-learning studies.

REFERENCES

[1] Aha, D.: Generalizing from case studies: A case study. Proceedings of the 9th International Conference on Machine Learning pp. 1–10 (1992)

[2] Ali, S., Smith, K.: On learning algorithm selection for classification. Applied Soft Computing Journal **6**(2), 119–138 (2006)

[3] Blockeel, H., Vanschoren, J.: Experiment databases: Towards an improved experimental methodology in machine learning. Lecture Notes in Computer Science **4702**, 6–17 (2007)

[4] Brain, D., Webb, G.: The need for low bias algorithms in classification learning from large data sets. PKDD '02: Proceedings of the 6th European Conference on Principles of Data Mining and Knowledge Discovery pp. 62—73 (2002)

[5] Brazdil, P., Giraud-Carrier, C., Soares, C., Vilalta, R.: Metalearning: Applications to data mining. Springer (2009)

[6] Caruana, R., Niculescu-Mizil, A.: An empirical comparison of supervised learning algorithms. Proceedings of the 23rd International Conference on Machine Learning (ICML'06) pp. 161–168 (2006)

[7] Chandrasekaran, B., Josephson, J.: What are ontologies, and why do we need them? IEEE Intelligent systems **14**(1), 20–26 (1999)

[8] Demsar, J.: Statistical comparisons of classifiers over multiple data sets. Journal of Machine Learning Research **7**, 1–30 (2006)

[9] Dietterich, T.: Approximate statistical tests for comparing supervised classification learning algorithms. Neural computation **10**(7), 1895–1923 (1998)

[10] Hand, D.: Classifier technology and the illusion of progress. Statistical Science (2006)

[11] Hilario, M., Kalousis, A., Nguyen, P., Woznica, A.: A data mining ontology for algorithm selection and meta-mining. Proceedings of the ECML/PKDD09 Workshop on 3rd generation Data Mining (SoKD-09) pp. 76–87 (2009)

[12] Horridge, M., Knublauch, H., Rector, A., Stevens, R., Wroe, C.: A practical guide to building owl ontologies using protege 4 and co-ode tools. The University of Manchester (2009)

[13] Hoste, V., Daelemans, W.: Comparing learning approaches to coreference resolution. there is more to it than bias. Proceedings of the Workshop on Meta-Learning (ICML-2005) pp. 20–27 (2005)

[14] Kalousis, A., Hilario, M.: Building algorithm profiles for prior model selection in knowledge discovery systems. Engineering Intelligent Systems **8**(2) (2000)

[15] Karapiperis, S., Apostolou, D.: Consensus building in collaborative ontology engineering processes. Journal of Universal Knowledge Management **1**(3), 199–216 (2006)

[16] Keogh, E., Kasetty, S.: On the need for time series data mining benchmarks: A survey and empirical demonstration. Data Mining and Knowledge Discovery **7**(4), 349–371 (2003)

[17] Kuehl, R.: Design of experiments: statistical principles of research design and analysis. Duxbury Press (1999)

[18] Michie, D., Spiegelhalter, D., Taylor, C.: Machine learning, neural and statistical classification. Ellis Horwood (1994)

[19] Nielsen, M.: The future of science: Building a better collective memory. APS Physics **17**(10) (2008)

[20] Panov, P., Soldatova, L., Dzeroski, S.: Towards an ontology of data mining investigations. Lecture Notes in Artificial Intelligence **5808**, 257–271 (2009)

[21] Perlich, C., Provost, F., Simonoff, J.: Tree induction vs. logistic regression: A learning-curve analysis. The Journal of Machine Learning Research **4**, 211–255 (2003)

[22] Pfahringer, B., Bensusan, H., Giraud-Carrier, C.: Meta-learning by landmarking various learning algorithms. Proceedings of the 17th International Conference on Machine Learning pp. 743–750 (2000)

[23] Salzberg, S.: On comparing classifiers: A critique of current research and methods. Data mining and knowledge discovery **1**, 1–12 (1999)

[24] Soldatova, L., King, R.: An ontology of scientific experiments. Journal of the Royal Society Interface **3**(11), 795–803 (2006)

[25] Stoeckert, C., Causton, H., Ball, C.: Microarray databases: standards and ontologies. nature genetics **32**, 469–473 (2002)

[26] Szalay, A., Gray, J.: The world-wide telescope. Science **293**, 2037–2040 (2001)

[27] Vanschoren, J., Blockeel, H., Pfahringer, B.: Experiment databases: Creating a new platform for meta-learning research. Proceedings of the ICML/UAI/COLT Planning to Learn Workshop, pp. 10–15 (2008)

[28] Vanschoren, J., Pfahringer, B., Holmes, G.: Learning from the past with experiment databases. Lecture Notes in Artificial Intelligence **5351**, 485–492 (2008)

Using Active Testing and Meta-Level Information for Selection of Classification Algorithms

Rui Leite [1] and **Pavel Brazdil** [1] and **Francisco Queirós** [2]

Abstract. The problem of selecting the best classification algorithm for a specific problem continues to be very relevant. The number of classification algorithms has grown significantly. Testing all alternatives is not really a viable option. If we compare pairs of algorithms, which was advocated by some researchers, the number of comparisons grows exponentially. To avoid this problem we suggest a method referred to as *active testing*. The aim is to reduce the number of comparisons by carefully selecting which tests should be carried out. The method described uses meta-knowledge concerning past experiments and proceeds in an iterative manner. In each iteration the best ranked algorithm is identified together with its best competitor. This determines the test to be carried out. The result of the test determines which algorithm should be eliminated from consideration. To speed up the process, we use a fast method that exploits information on partial learning curves and just predicts which algorithm is better. The method terminates when there are no more alternatives to be explored. The method was evaluated in a leave-one-out fashion. The results show that the method is indeed effective in determining quite accurately the best algorithm using a limited number of tests.

1 Introduction

The problem of selecting the best classification algorithm for a specific problem continues to be very relevant. The number of classification algorithms has grown significantly. Moreover, the performance of some classification algorithms is very sensitive to parameter settings. Therefore the issue of algorithm selection (or in general KDD workflows) has been an object of study in many works during the past 20 years (see e.g. [16, 3, 22, 19, 18]). Most approaches rely on the concept of *metalearning*. This approach exploits information concerning characterization of datsets and their past performance to guide the choice regards which algorithm(s) should be used on the current dataset. The term *metalearning* comes from the fact that we try to learn the function that maps *problem / dataset characterization* to *performance estimate* and then applies this to a new problem / dataset. The dataset characterisation is often captured in the form of various statistical and information–theoretic measures [16, 3].

Others have advocated the use *sampling landmarks*, representing perfomance results of algorithms on subsets of data [20, 8]. These can be used to characterize the given datasets as other measures do. A series of sampling landmarks represents in effect a partial learning curve. In a subsequent work [11, 12] it was shown that use of sampling landmarks leads to substantially better results than previous approaches that exploit just classical dataset characteristics. Moreover, it was shown that it is possible to combine *sampling landmarks* with classical dataset characteristics which may represent the best of the two alternatives [12].

However, it has been argued that the characterization of classification algorithms is sensitive to which pair of algorithms we have in mind [1]. This is true also when considering sampling landmarks. Therefore some authors have focused on a particular subtask of how to select the appropriate algorithm from a pair. However as usually we consider more that two algorithms we must have some method to decide which algorithm is likely to be the best one. Some authors have proposed that a set of paiwise statistical tests be carried out and then use some aggregate measure (such as the overall number of "wins") for each algorithm. This measure is used to identify the best candidate, or alternatively, construct a ranking of candidate algorithms. This strategy was followed, for instance, in [1] and recently in [13]. The disadvantage of this approach is that the number of binary tests grows exponentially with the number of algorithms. To overcome this limitation we suggest a method that is referred to as *active testing* which bears some similarities to *active learning*. The aim is to reduce the number of comparisons by carefully selecting the tests to be carried out.

The method described uses meta-knowledge concerning past experiments and proceeds in an iterative manner. In each iteration the best ranked algorithm is identified together with its best competitor. This determines the test to be carried out. The result of the test determines which algorithm should be eliminated from consideration. To speed up the process, we use a fast method that exploits information on partial learning curves mentioned before [12] which predicts which algorithm is better. The method terminates when there are no more alternatives to be explored.

The method was evaluated in a leave-one-out fashion. The results show that the method is indeed effective in determining quite accurately the best algorithm using a limited number of tests. The advantage of this method is that there is no need to conduct all possible tests involving pairs of algorithms.

We also show that output of the method generated on different datasets can be integrated to obtain a kind of *contingency plan*, that provides an effective plan of tests to be carried out in a new situation (i.e. for a given dataset).

The paper is organized as follows. In Section 2 we discuss some underlying assumptions of this work. In Section 3 we describe the proposed method in outline. Subsections 3.1 till 3.4 give more details about each step. In one of its sections we review the method for determining which of the two given algorithms is better. This method exploits sampling landmarks and data characteristics. Section 4 dis-

[1] LIAAD-INESC Porto L.A./Faculty of Economics, University of Porto, Portugal, email: rleite@liaad.up.pt, pbrazdil@liaad.up.pt

[2] Faculty of Economics, University of Porto, Portugal, email: 080401175@fep.up.pt

cusses evaluation and presents the results. The final section presents discussion, future work and the conclusions.

2 Underlying Assumptions

In this section we describe some underlying assumptions of this work. These were then put to test, by first implementing the method and then by conducting experiments.

2.1 Metadata provides probability estimates

Given a set of classification algorithm with given parameter settings and some new classification problem (dataset d_{new}) the aim is to identify the potentially best algorithm for this task with respect to some given performance measure (e.g. accuracy, AUC, rank etc.). Let us represent the performance of algorithm a_i on dataset d_{new} using $M(a_i, d_{new})$. So our aim is to identify an algorithm $a*$, for which the performance measure is maximum. In other words, we seek an algorithm that maximizes the performance gain over other competitors. Maximizing a performance gain is of course equivalent to minimizing a potential loss. The algorithm $a*$ should satisfy the following constraint:

$$\forall a_i M(a*, d_{new}) \geq M(a_i, d_{new}) \tag{1}$$

As our estimate $\hat{a*}$ might not be perfect, we are interested in the probability

$$P(M(\hat{a*}, d_{new}) \geq M(a_i, d_{new})) \tag{2}$$

We want this probability to be as high as possible, ideally near to 1. The decision concering $\geq$ may be established using a statistical test or even using a simple comparison.

As in other work that exploits metalearning, we will assume that the probabality estimate can be obtained by considering similar past cases. Let Ds represent a subset of all datasets D that is similar to d_{new} (later on we will explain how such similar datasets can be identified). The assumption that is being explored here is the following:

$$P(M(\hat{a*}, d_{new}) \geq M(a_i, d_{new})) \sim P(M(\hat{a*}, Ds) \geq M(a_i, Ds)) \tag{3}$$

That is, the subset of datasets Ds is used to estimate this probability. For a fixed Ds and a particular a_i (e.g. the best current guess) the latter estimate can be done quite easily. We simply go though all algorithms (except a_i) and identify the one that satisfies the constraint in a maximum proportion of cases. Laplace correction is used to avoid obtaining high estimates based on very few cases. So the objective is:

$$\frac{\arg\max_{a_k} \sum_{dj \in Ds} (i(M(a_k, d_j) \geq M(a_i, d_j))) + 1}{|Ds| + 2} \tag{4}$$

where $|Ds|$ represents the size of Ds and $i(test)$ an identity function whose value is 1, if the $test$ is true and 0 otherwise.

2.2 Some considerations concering similarity

Let us now see how the subset of dataset Ds can be identified for a given d_{new}. This could be done with the help of various statistical and information–theoretic measures that were mentioned earlier [16, 3]. Another possibility is to use tests already carried out on a new dataset. Suppose, we have carried out a test with a_1 and a_2 and obtained, say, the result $M(a_2, d_{new}) \geq M(a_1, d_{new})$. The result of this test can be used as a characteristic. It can be used to identify those datasets where a similar situation occurred in the past (i.e. a_2 achieved a better result than a_1, shortly $a_2 > a_1$). It is assumed that this notion is relevant for our method whose aim is to identify the best algorithm for the new dataset.

2.3 Role of active learning

Another assumption underlying this work is that estimates of probability can be explored to select a test to be carried out. Selecting a test can be compared to a selection of an example to be labeled in active learning. We recall that the result of the test is used to identify Ds, so here active learning is more concerned with getting an informative feature value, rather than classifying an example. The assumption adopted here is that if the probability of a_j winning over a_i is high then this test is *informative* and is a good candidate to be carried out.

2.4 Other work on active learning

Quite a lot of work has been done on experiment design [6] and active learning. Our method described here follows the main trends that have been outlined in literature. The aim of active learning is to reduce labeling effort in supervised learning, usually in classification contexts [15].

There is relatively little work on active learning for ranking tasks. One notable exception is [14], who use the notion of Expected Loss Optimization (ELO). Another work that is relevant is [4]. Their aim was to identify best substances for drug screening using a minimum number of tests. In the experiments described, the authors have focused on top-10 substances. Several different strategies were considered and evaluated.

Our problem here is not ranking, but rather finding the potentially the best element, so this work is only partially relevant.

3 Using Active Testing and Meta-Level Information for Selection of Classification Algorithms

In this section we describe a method AT that is based on the assumptions discussed in the previous section. The aim is to identify the best classification algorithm for a given dataset. The method involves the following steps:

1. Construct a ranking of a given set of algorithms using information from past experiments (metadata).
2. Identify the best algorithm in the ranking (the top-ranked algorithm).
3. Find the most promising competitor of the best algorithm.
4. Conduct a test (preferably some fast test) to determine which of the two algorithms is better.
   Eliminate the algorithm that was worse in the pairwise test from further consideration (i.e. from the ranking).
5. Repeat the whole process starting with step 2 until no competitor can be identified. If this occurs output the algorithm representing the overall winner.

The following subsections present more details concerning each step.

3.1 Establishing Ranks of Algorithms Using Metadata

It is assumed that information concerning past experiments is available and that it includes performance results of a given set of classification algorithms on different datasets. This kind of information is normally referred to as *metadata*.

This performance information can be expressed by *success rate (accuracy)* or its corresponding rank, *AUC* (area under the ROC curve), or any other suitable measure of classifier performance.

Here we will use *success rate (accuracy)* and their ranks. In this paper we describe a small-scale study which involves 6 different classification algorithm from Weka [9] (IB1, J48, JRIp, LogD, MLP and NB) on 40 UCI datasets [2]. The fact that we selected some algorithms, but not others (e.g. SVM) is irrelevant for the problem studied. A large-scale study is underway in collaboration with J.Vaschoren. It exploits about 290 algorithms from experimental database ([10],[21]).

Table 1 shows performance results of our small-scaled study. It involves accuracies of different classification algorithm referred to above. Each accuracy figure represents a mean of 10 values obtained in 10-fold cross-validation. The ranks of algorithms are shown in brackets just after the accuracy value. So, for instance, if we consider dataset $abalone$, algorithm MLP is attributed rank 1 as its accuracy is highest on this problem. The second rank is occupied by $LogD$ etc.

We recall that the aim of step 1 it to construct a ranking of all algorithms. This is done in twi steps. In the first step the performance results on each dataset are analysed and ranks attributed to each algorithm. In the second step the individual rank values are aggregated by calculating the *mean rank* of each algorithm. The mean rank of algorithm aj, represented by R_{aj}, is $\frac{1}{n}\sum_{di=1}^{n} R_{aj,di}$ where $R_{aj,di}$ represents the rank of algorithm aj on dataset di and n is the number of datasets. This is a quite common procedure often used in machine learning to assess how a particular algorithm compares to the others (see e.g. [5]).

Table 1. Ranking of Algorithms and Mean Rank

Datasets	IB1		J48		JRip		LogD		MLP		NB	
abalone	.197	(5)	.218	(4)	.185	(6)	.259	(2)	.266	(1)	.237	(3)
acetylation	.844	(1)	.831	(2)	.829	(3)	.745	(5)	.609	(6)	.822	(4)
adult_metal	.794	(6)	.861	(1)	.843	(3)	.850	(2)	.830	(5)	.834	(4)
allbp	.964	(5)	.973	(2)	.975	(1)	.966	(4)	.970	(3)	.942	(6)
allhyper	.975	(5)	.988	(1)	.985	(2)	.979	(4)	.981	(3)	.954	(6)
ann	.924	(6)	.997	(1)	.995	(2)	.959	(4)	.971	(3)	.954	(5)
byzantine	.990	(2)	.952	(5)	.936	(6)	.986	(3)	.991	(1)	.980	(4)
car	.778	(6)	.931	(3)	.882	(4)	.935	(2)	.992	(1)	.855	(5)
cmc	.428	(6)	.527	(2)	.523	(3)	.506	(4)	.532	(1)	.498	(5)
contraceptive	.428	(6)	.527	(2)	.523	(3)	.506	(4)	.532	(1)	.498	(5)
injury_severity	.770	(6)	.839	(2)	.827	(3)	.813	(4)	.845	(1)	.771	(5)
internetad	.960	(4)	.967	(2)	.968	(1)	.951	(5)	NA	(6)	.966	(3)
isolet	.897	(2)	.841	(4)	.787	(5)	NA	(6)	.970	(1)	.851	(3)
krkopt	.372	(5)	.564	(2)	.395	(4)	.405	(3)	.635	(1)	.360	(6)
krvskp	.900	(5)	.994	(2)	.991	(3)	.976	(4)	.994	(1)	.877	(6)
led24	.482	(6)	.716	(3)	.701	(4)	.725	(1)	.681	(5)	.724	(2)
letter	.959	(1)	.881	(2)	.859	(3)	.774	(5)	.827	(4)	.642	(6)
mfeat	.978	(2)	.940	(4)	.920	(5)	NA	(6)	.984	(1)	.953	(3)
musk	.957	(4)	.970	(2)	.961	(3)	.952	(5)	1.000	(1)	.838	(6)
nursery	.776	(6)	.970	(2)	.967	(3)	.925	(4)	.998	(1)	.902	(5)
optdigits	.988	(1)	.902	(6)	.904	(5)	.939	(3)	.984	(2)	.914	(4)
page	.958	(5)	.968	(2)	.970	(1)	.965	(3)	.961	(4)	.898	(6)
parity	.508	(3)	.779	(2)	.449	(4)	.371	(5)	.922	(1)	.368	(6)
pendigits	.994	(1)	.962	(3)	.963	(2)	.956	(4)	.945	(5)	.857	(6)
pyrimidines	.946	(3)	.946	(2)	.936	(4)	.907	(5)	.959	(1)	.822	(6)
quadrupeds	1.000	(3)	1.000	(6)	1.000	(3)	1.000	(3)	1.000	(3)	1.000	(3)
quisclas	.577	(5)	.622	(3)	.644	(2)	.694	(1)	.578	(4)	.396	(6)
reclJan2jun97	.942	(5)	.946	(4)	.948	(3)	.949	(2)	.949	(1)	.850	(6)
sat	.901	(1)	.861	(4)	.863	(3)	.858	(5)	.893	(2)	.795	(6)
segmentation	.974	(1)	.969	(2)	.960	(3)	.957	(5)	.958	(4)	.800	(6)
shuttle	.999	(3)	1.000	(1)	1.000	(2)	.968	(5)	.997	(4)	.930	(6)
sick	.961	(5)	.986	(1)	.985	(2)	.969	(4)	.970	(3)	.928	(6)
spambase	.907	(4)	.932	(1)	.926	(3)	.927	(2)	.895	(5)	.797	(6)
splice	.749	(6)	.941	(3)	.932	(4)	.907	(5)	.956	(1)	.954	(2)
taska_part_hhold	.533	(4)	.803	(1)	.801	(2)	NA	(5.5)	NA	(5.5)	.578	(3)
thyroid0387	.817	(4)	.958	(1)	.941	(2)	NA	(6)	.849	(3)	.783	(5)
triazines	.938	(2)	.920	(3)	.905	(4)	.755	(5)	.947	(1)	.677	(6)
waveform21	.770	(5)	.762	(6)	.796	(4)	.869	(1)	.844	(2)	.809	(3)
waveform40	.738	(6)	.741	(5)	.792	(4)	.866	(1)	.834	(2)	.801	(3)
yeast	.526	(6)	.562	(5)	.577	(3)	.589	(2)	.591	(1)	.573	(4)
mean rank	4.05		2.73		3.17		3.74		2.54		4.78	

In Table 1 the mean ranks are shown at the bottom of the table. These values permit us to obtain the final ordering of algorithms, O. In our case it is $O = \langle MLP, J48, JRIp, LogD, IB1, NB\rangle$.

3.2 Identifying the Current Best Algorithm and the Most Promising Competitor

The final ordering of algorithms, O, established in the previous step permits to identify the best candidate algorithm, which is the one that appears in the first position. Let us identify this algorithm as a_{best} Considering the values shown earlier in Table 1 the best algorithm is $a_{best} = MLP$. However, we do not stop here. we need to identify also its *best competitor*. To identify the best competitor, all the algorithms in the list O are considered, one by one, starting at position 2. For each one we analyse the information of past experiments (meta-level information) to determine its chances of 'winning' over a_{best} (i.e. achieving higher performance than a_{best}). We seek one that has the highest chances of winning, following the strategy described in the previous section. The most promissing competitor is the one that satisfies the following expression:

$$\frac{\arg\max_{a_k} \sum_{dj \in Ds}(i(M(a_k, d_j) \geq M(a_{best}, d_j))) + 1}{|Ds| + 2} \quad (5)$$

Should there be two or more algorithms that have the same probability winning over a_{best}, the competitor that appears on the first places in ranking O is chosen. That is, we follow a hill-climbing approach.

In the expression above the term rel-datasets represents *relevant datasets*. These are initially all datasets, but later, as different tests have been carried out, *relevant datasets* is a certain subset for which certain conditions get verified. But we come to this point later.

3.3 Conducting Tests on a Pair of Algorithms

This step involves carrying out a test which includes the best algorithm a_{best} (determined so far) and its best competitor $a_{\text{best-comp}}$ on the new dataset.

In our experiments we simply retrieve the results from a meta-database that we have constructed before. If, however, the dataset were really a new one, we would need to carry out a test that includes both algorithms on this dataset. We could use any classifier evaluation procedure in this step. It could be cross-validation procedure, or some faster method. Here we have opted for a relatively fast method based on our previous work [12], which just gives an estimate concerning which algorithm is better out of the two. The method is explained briefly in the last part of this section.

After the test has been carried out, one of the algorithms (the worse one) is eliminated from the list of ranked algorithms. Furthermore, the test is stored for further reference as *dataset selection context*. It affects the focus on relevant datasets to be used in the process of searching for competitors.

Suppose the test a_i versus a_j was carried out and the result was $a_i > a_j$. Here the shorthand $a_i > a_j$ is used to indicate that a_i was better than a_j. The relevant datasets are those where the same condition is verified (i.e. $a_i > a_j$).

The process of identifying the best algorithm and its best competitor for subsequent test is referred to here as *active testing*. We use this term to draw attention to the fact that this strategy shares some similarities to *active learning* [7] in machine learning. The concern of both active testing and active learning is to speed up the process

of improving a given performance measure. In active learning this is done by selecting one or more unlabeled examples that are considered to be most informative should the class be known. These are presented to the user and labelled. In active testing this is done by selecting a test that is considered the most beneficial should its result be known. This test is executed in order to obtain its result.

Note that the test $a_i \; versus \; a_j$ can be considered as a classification problem. If a_i wins over a_j we obtain one class value (here we use 1). If it loses, we obtain a different class value (here -1). If the result is a draw, we obtain yet another class value (here 0).

3.3.1 *Fast method of estimating which one of two algorithms is better*

In this part we give an overview of the method that estimates which one of the two given algorithms (a_i, a_j) is more likelly to be better on a given dataset. This method is referred to as SAM (Selection of Algorithms using Meta-learning). More details concerning this method can be found in [11, 12, 13].

As has been observed in the previous subsection the problem of determining which one of the two given algorithms is better can be regarded as a classification problem. The method uses both data characteristics and information about accuracies of the two classifications algorithms on samples of data for different datasets. This information includes:

- Information about various datasets used in the past (meta-database).
- Information about the new dataset.

As has been observed before the performance of the two algorithms on a set of samples sheds some light on the final outcome of the learning. Fig. 1 illustrates this. Its shows the measured accuracy on three samples of data s_1, s_2 and s_3. Here $s_1 \cdots s_m$ are used to represent random samples of increasing size.

The two curves have been extended on the basis of other similar curves retrieved from the metadatabase. It enables to determine that algorithm A_q will achieve higher accuracy at the end, that is, when the full set of examples has been used.

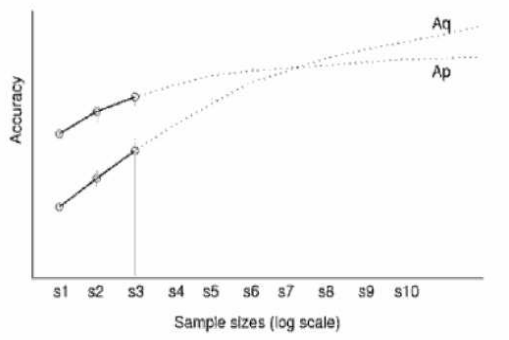


Figure 1. Example of two extended learning curves determining which algorithm is better

The method SAM is divided in two subroutines: SAM_F and $SetGen$. Subroutine SAM_F predicts which algorithms is better using a fixed set of *meta-features* indicated as parameters. These meta-features represent both general data characteristics like *number of cases*, *class entropy*, etc. and also known points of *partial learning curves*.

Subroutine $SetGen$ searches for a subset of the best meta-features satisfying two criteria: (1) maximize the improvement the performance SAM_F (accuracy in deciding which of the two algorithms is better), while (2) trying to reduce the costs involved in obtaining the values of meta-features (computing a point on the learning curve requires conducting a test, i.e. initiating learning and evaluating the model). Therefore we attend to both criteria.

This method also exploits the active testing strategy discussed in the previous subsection.

Subroutine SAM_F requires as input a fixed set of samples for each algorithm. Let S_i (S_j) represent the samples required to characterize algorithm a_i (a_j). So, for instance, the samples passed as input can be $S_i =< s_1, s_2, s_3 >$ and $S_j =< s_1, s_2 >$. The method encompasses the following steps:

1. Compute the data characteristics for the new dataset d.
2. Characterize the new dataset d by conducting experiments with algorithm a_i on S_i samples and measuring accuracies. Repeat this for a_j on S_j samples. In other words build two partial learning curves.
3. Compute the distances between the information relative to dataset d and stored information relative to all other datasets $d_1 \cdots d_n$.
4. Identify the subset of k nearest datasets.
5. For each of the k nearest datasets identified and retrieved in the previous step, *adapt* each pair of learning curves to the new partial learning curves build for dataset d. Adaptation is done by rescaling each retrieved learning curve in order to minimize the square distance from this curve to the respective partial learning curve for dataset d.
6. For each pair of adapted curves decide which algorithm achieves higher performance on the adapted and extended learning curves.
7. Identify the algorithm that is better on the new dataset d, by considering the results on k pairs of nearest datasets.

Method SAM was evaluated by the authors using the leave-one-out methodology discussed in Section 3. The results have shown that the method was about 90% accurate in predicting the best algorithm. The mean runtime of the method when compared to selection using cross-validation was about 7 times lower.

3.4 Repeating the Method and Stopping Criteria

The whole process of identifying the best algorithm and its best competitor (steps 2 and 3) and conducting tests (step 4) is repeated until no competitor can be identified. This process is controlled by list O. If the list contains just one item (i.e. algorithm), the process stops. The method outputs the algorithm that has successfully defended itself against all competitors. This algorithm can be regarded as the overall winner.

4 Evaluation and Results

4.1 Leave-one-out Evaluation

The evaluation of the proposed method was done using so called *leave-one-out* method [17]. The experiments reported here involve N (40) datasets and so the whole procedure was repeated N (40) times. In each cycle one dataset was left out for testing and the remaining $N - 1$ (39) datasets were used to determine the best candidate algorithm.

As we know what the best algorithm is, we can establish the rank of the candidate algorithm that was predicted by our method. Our criteria of success is the following. The value should be better than the rank of best performing algorithm overall. In our small-scale experiment this is MLP that achieved a mean rank of 2.54.

Table 2. Rank of predictions / real / default

Dataset	test AT_{full}	rank	test $AT_{fast/full}$	rank	test AT_{fast}	rank	rank default	true best
abalone	MLP	1	MLP	1	MLP	1	1	MLP
acetylation	IB1	1	J48	2	J48	2	6	IB1
adult_metal	J48	1	JRIp	3	JRIp	3	5	J48
allbp	JRIp	1	J48	2	J48	2	3	JRIp
allhyper	J48	1	JRIp	2	JRIp	2	3	J48
ann	J48	1	J48	1	J48	1	3	J48
byzantine	MLP	1	IB1	2	IB1	2	1	MLP
car	MLP	1	MLP	1	MLP	1	1	MLP
cmc	MLP	1	J48	2	J48	2	1	MLP
contraceptive	MLP	1	J48	2	J48	2	1	MLP
injury_severity	MLP	1	J48	2	J48	2	1	MLP
internetad	JRIp	1	JRIp	1	JRIp	1	6	JRIp
isolet	MLP	1	MLP	1	NB	3	1	MLP
krkopt	MLP	1	LogD	3	LogD	3	1	MLP
krvskp	MLP	1	J48	2	J48	2	1	MLP
led24	J48	3	J48	3	J48	3	5	LogD
letter	IB1	1	MLP	4	MLP	4	4	IB1
mfeat	MLP	1	IB1	2	IB1	2	1	MLP
musk	MLP	1	MLP	1	MLP	1	1	MLP
nursery	MLP	1	MLP	1	MLP	1	1	MLP
optdigits	IB1	1	IB1	1	IB1	1	2	IB1
page	JRIp	1	JRIp	1	JRIp	1	4	JRIp
parity	MLP	1	MLP	1	MLP	1	1	MLP
pendigits	JRIp	2	IB1	1	IB1	1	5	IB1
pyrimidines	MLP	1	MLP	1	MLP	1	1	MLP
quadrupeds	MLP	3	IB1	3	IB1	3	3	IB1
quisclas	JRIp	2	JRIp	2	JRIp	2	4	LogD
recljan2jun97	MLP	1	LogD	2	LogD	2	1	MLP
sat	IB1	1	IB1	1	IB1	1	2	IB1
segmentation	IB1	1	IB1	1	IB1	1	4	IB1
shuttle	J48	1	JRIp	2	JRIp	2	4	J48
sick	J48	1	J48	1	J48	1	3	J48
spambase	J48	1	J48	1	J48	1	5	J48
splice	MLP	1	MLP	1	MLP	1	1	MLP
taska_part_hhold	J48	1	JRIp	2	LogD	6	5.5	J48
thyroid0387	J48	1	J48	1	NB	5	3	J48
triazines	MLP	1	J48	3	J48	3	1	MLP
waveform21	LogD	1	LogD	1	LogD	1	2	LogD
waveform40	LogD	1	LogD	1	LogD	1	2	LogD
yeast	MLP	1	LogD	2	LogD	2	1	MLP
Mean Rank		1.15		1.68		1.92	2.54	

The results of our method, referred to as AT_{fast} are shown in Table 2. The results are compared to two somewhat slower variants, which differ from the method described in how pairwise tests are carried out.

We note that fast tests based on partial learning curves do not always return a result. In other words, the method does not always provide an estimate regards which of the two algorithms is better on a given dataset. This happens, for instance, when a slow algorithm does not return a result within in a pre-established time allocation. Fortunately, this problem is not very frequent. In our experiments it occurred only in 5 datasets out of 40. Still, it is necessary to have some way of deciding what to do in such cases. So the first variant, referred to as $AT_{fast/full}$ in Table 2 uses a slower test, based on cross-validation (CV), in such situations.

The second variant uses a slower test based on cross-validation in *all* situations. This variant is referred to as AT_{full} in Table 2.

The results of our method (AT_{fast}) show that the mean rank of our fast method is 1.92, which we consider a good result, as it is much better than the rank of the best-peforming algorithm in our small-scale experiments, which was MLP with mean rank of 2.54. Our method surpasses this performance by quite a good margin.

The method referred to as $AT_{fast/full}$ in Table 2 achieved better mean rank (1.68) than the fast method. This is not surprising, as in all situations when SAM did not return a result we have conducted a CV test.

Finally, the method referred to as AT_{full} in Table 2 achieved a mean rank of 1.15 which is very near to the ideal value. This is an excellent result.

4.1.1 Analysis of number of steps needed

Let us analyse the times of the fast method (AT_{fast}) proposed here. Our method is many times faster than the method that would use CV to select the best algorithm, identified as *true best* in Table 2. The first saving comes from using active testing strategy. As in our set-up with 6 algorithms, the number of all possible pairwise tests is $\binom{6}{2} = 6 * 5/2 = 15$. We have analysed the average number of tests needed to reach a decision. The number is 3 (see the nest subsection), that is $1/5$ of all possible tests. The fast tests are also about 7 times faster that tests that involve CV [12]. If we were to use CV to select the best algorithm for the new dataset, we would of course not need to run pairwise tests, but simply run CV for the 6 algorithms. Still, the time saving is quite substantial.

The method referred to as AT_{full} in Table 2 achieved a very good mean rank in small-scale experiments, but we need to be aware that this method is not so fast. It requires 3 CV tests, each involving 2 algorithms. As one of the algorithms is common in two subsequent tests, we need to evaluate 4 algorithms using a CV, instead of 6.

However, we note that the number of tests grows approximately with $log(|A|$, where $|A|$ represents the number of algorithms (and their different variants). So if the number of algorithms were, say, 128, we expect, that the number of tests needed would not exceed $log_2(128) = 7$ by much. We have conducted preliminary tests with 290 algorithms, which confirmed that this estimate is more or less correct. The detailed results will be presented in a future paper.

4.2 Generating a Contigency Plan

Suppose the method presented here is used with all datasets and all recommended tests get recorded. As some of tests can then be merged (here we have used a manual process) the resulting structure appears in the form of a tree. The resulting tree is shown in Fig. 2. However, it would not be difficult to automate fully this process.

We note that the tree starts with the most important test to be carried out which in our case is $MLP\ versus\ J48$. The result of this test determines what needs to be done next. Note that we do not know beforehand what the result will be. Still the figure contemplates both alternatives (as both were used in the past). This is why we refer to the tree as a *contingency plan*. It provides a plan regard what should be done in different situations (e.g. when $MLP < J48$ etc.).

Note that the contingency plan exploits metaknowledge, but reorganizes it in the form of an explicit plan that can be directly followed in a new situation either by the user or by the system.

5 Discussion, Future Work and Conclusions

5.1 Discussion and Future Work

As the method presented here is based on mean ranks, it is interesting to analyse the result of Bonferroni-Dunn test [5] which is an appropriate statistical test when multiple algorithms are being compared on different datasets. The result of this test is shown in Fig. 3. The horizontal blue line shows a critical distance (CD) for $p = 0.1$. The analysis of this figure confirms that the best algorithm is indeed MLP and that the best competitor in the first step is $J48$.

This work can be extended in several different directions. First, it would be interesting to use more algorithms (including parameterized versions) and more datasets. We plan to use the experimental database [10] for this purpose.

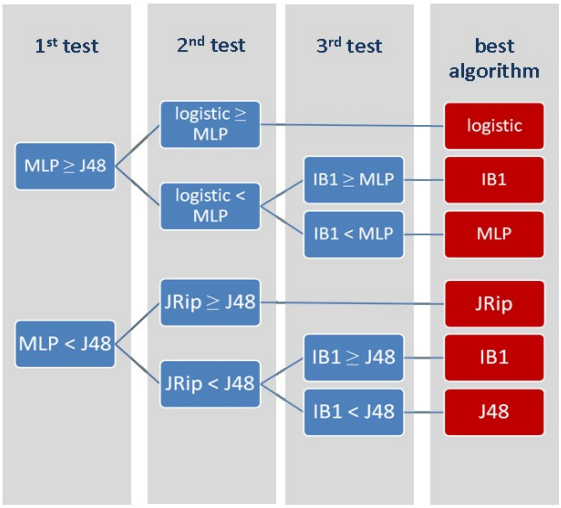


Figure 2. Contigency Plan

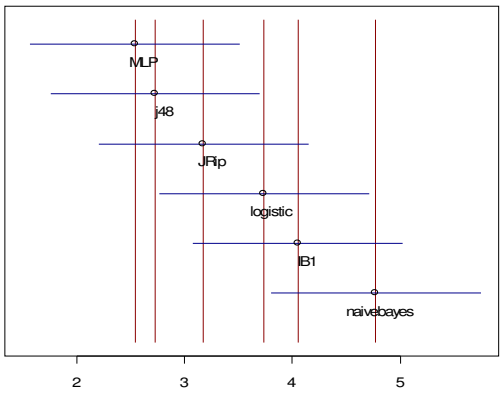


Figure 3. Results of Bonferroni-Dunn test relative to the initial situation

5.2 Conclusions

The problem of selecting the best classification algorithm for a specific problem continues to be a very relevant issue. Testing all alternatives is not really a viable option. If we compare pairs of algorithms, which was advocated by some researchers, the number of comparisons grows exponentially. To avoid this problem we have suggested a method referred to as *active testing*. The aim is to reduce the number of comparisons by carefully selecting which tests should be carried out.

The method described (AT_{fast}) uses meta-knowledge concerning past experiments and proceeds in an iterative manner. In each iteration the best ranked algorithm is identified together with its best competitor. This determines the test to be carried out. The result of the test determines which algorithm should be eliminated from consideration. To speed up the process, we use a fast method that exploits information on partial learning curves and just estimates which algorithm is better. The method terminates when there are no more alternatives to be explored.

The method was evaluated in a leave-one-out fashion. The results show that the method is indeed effective in determining quite accurately the best algorithm using a limited number of tests. In our experimental setting we have used 6 classification algorithms with 40 datasets.

Our results in small-scale experiments show that the mean rank of our fast method is 1.92, which we consider to be a good result, as it is much better than the mean rank of the best performing algorithm, that is, MLP with a mean rank of 2.54.

This method (AT_{fast}) was compared to two other alternatives. The first variant, referred to as $AT_{fast/full}$ uses a slower test, based on cross-validation (CV), in situations when the fast test has not returned a result. The mean rank of this method is 1.68. The second variant uses a slower test based on cross-validation in *all* situations. This variant is referred to as AT_{full}. The mean rank of this method is 1.15 which is very near to the ideal value.

As in our setting with 6 algorithms only, this method does not save much time (about $1/3$). However, we have argued that if the number of algorithms and its variants (N) were larger, the saving could be substantial. We expect that the number of tests needed would not exceed much $log(N)$. This was confirmed by preliminary experiments with 290 algorithms.

Finally, we have shown that the output of the method on all datasets can be re-arranged in the form of a tree, representing in effect a *contingency plan*. It provides a plan regard what should be done in different situations (e.g. when $MLP < J48$ etc.). The contingency plan exploits metaknowledge, but reorganizes it in the form of explicit plan that can be directly followed. We consider that this is another important result which can be of general use to the machine learning community.

Acknowledgements

The authors with to acknowledge the support under grant BII-2009 awarded to Francisco Queirós and also the pluri-annual support provided by FCT to LIAAD-INESC Porto L.A.

REFERENCES

[1] A.Kalousis and M.Hilario, 'Feature selection for metalearning', in *D.Cheung et al.(eds): Proc. of the Fifth Pacific–Asia Conf. on Knowledge Discovery and Data Mining*. Springer, (2001).

[2] C.L. Blake and C.J. Merz. UCI repository of machine learning databases. http://www.ics.uci.edu/∼mlearn/mlrepository.html, 1998.

[3] P. Brazdil, C. Soares, and J. Costa, 'Ranking learning algorithms: Using IBL and meta-learning on accuracy and time results', *Machine Learning*, **50**, 251–277, (2003).

[4] K. De Grave, J. Ramon, and L. De Raedt, 'Active learning for primary drug screening', in *Proceedings of Discovery Science*. Springer, (2008).

[5] J. Demsar, 'Statistical comparisons of classifiers over multiple data sets', *The Journal of Machine Learning Research*, **7**, 1–30, (2006).

[6] V. Fedorov, *Theory of Optimal Experiments*, Academic Press, New York, 1972.

[7] Y. Freund, H. Seung, E. Shamir, and N. Tishby, 'Selective sampling using the query by committee algorithm', *Machine Learning*, **28**, 133–168, (1997).

[8] Johannes Fürnkranz and Johann Petrak, 'An evaluation of landmarking variants', in *Proceedings of the ECML/PKDD Workshop on Integrating Aspects of Data Mining, Decision Support and Meta-Learning (IDDM-2001)*, pp. 57–68. Springer, (2001).

[9] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten, 'The WEKA data mining software: an update', *SIGKDD Explor. Newsl.*, **11**(1), 10–18, (2009).

[10] H.Blockeel, 'Experiment databases: A novel methodology for experimental research', in *Leacture Notes on Computer Science 3933*. Springer, (2006).

[11] Rui Leite and Pavel Brazdil, 'Predicting relative performance of classifiers from samples', in *ICML '05: Proceedings of the 22nd international conference on Machine learning*, pp. 497–503, New York, NY, USA, (2005). ACM Press.

[12] Rui Leite and Pavel Brazdil, 'An iterative process for building learning curves and predicting relative performance of classifiers', in *Progress In Artificial Intelligence, Proceedings of the 13th Portuguese Conference on Artificial Intelligence Workshops (EPIA 2007)*, eds., J.Neves, M.F.Santos, and J.Machado, volume 4874 of *Lecture Notes in Computer Science*, pp. 87–98, Guimarães, Portugal, (December 2007). Springer.

[13] Rui Leite and Pavel Brazdil, 'Active testing strategy to predict the best classification algorithm via sampling and metalearning', in *Proceedings of the 19th European Conference on Artificial Intelligence - ECAI 2010 (to appear)*, (2010).

[14] B. Long, O. Chapelle, Y. Zhang, Y. Chang, Z. Zheng, and B. Tseng, 'Active learning for rankings through expected loss optimization', in *Proceedings of the SIGIR'10*. ACM, (2010).

[15] A. McCallum and K. Nigam, 'Employing EM and pool-based active learning for text classification', in *Proceedings of the 5th Int. Conf. on Machine Learning*, pp. 359–367, (1998).

[16] D. Michie, D.J.Spiegelhalter, and C.C.Taylor, *Machine Learning, Neural and Statistical Classification*, Ellis Horwood, 1994.

[17] Tom M. Mitchell, *Machine Learning*, McGraw-Hill, New York, 1997.

[18] John R. Rice, 'The algorithm selection problem', volume 15 of *Advances in Computers*, 65 – 118, Elsevier, (1976).

[19] Kate A. Smith-Miles, 'Cross-disciplinary perspectives on meta-learning for algorithm selection', *ACM Comput. Surv.*, **41**(1), 1–25, (2008).

[20] Carlos Soares, Johann Petrak, and Pavel Brazdil, 'Sampling-based relative landmarks: Systematically test-driving algorithms before choosing', in *Proceedings of the 10th Portuguese Conference on Artificial Intelligence (EPIA 2001)*, pp. 88–94. Springer, (2001).

[21] J. Vanschoren and H. Blockeel, 'A community-based platform for machine learning experimentation', in *Machine Learning and Knowledge Discovery in Databases, European Conference, ECML PKDD 2009*, volume LNCS 5782, pp. 750–754. Springer, (2009).

[22] Ricardo Vilalta and Youssef Drissi, 'A perspective view and survey of meta-learning', *Artif. Intell. Rev.*, **18**(2), 77–95, (2002).

Constraint Programming for Data Mining

Luc De Raedt

Dept. of Computer Science Katholieke Universiteit Leuven (Belgium),
luc.deraedt@cs.kuleuven.be <mailto:luc.deraedt@cs.kuleuven.be>

(Invited Talk)

Abstract

In this talk I shall explore the relationship between constraint-based mining and constraint programming. In particular, I shall show how the typical constraints used in pattern mining can be formulated for use in constraint programming environments. The resulting framework is surprisingly flexible and allows one to combine a wide range of mining constraints in different ways. The approach is implemented in off-the-shelf constraint programming systems and evaluated empirically. The results show that the approach is not only very expressive, but also works well on complex benchmark problems, sometimes even outperforming state-of-the-art data mining systems.

In addition to providing a detailed account of our actual initial results for item-set mining, I shall also argue that the use of constraint programming techniques and methodologies provides a new and interesting paradigm for data mining.

* The work reported is joint work with Tias Guns and Siegfried Nijssen.

References

1. http://dtai.cs.kuleuven.be/CP4IM/ for systems and publications.
2. De Raedt, L., Guns, T., and Nijssen, S. (2008). Constraint programming for itemset mining. *In Proc. of the SIGKDD.*
3. Nijssen, S., Guns, T., and De Raedt, L. (2009) Correlated Itemset Mining in ROC Space: A Constraint Programming Approach. *In Proc. of the SIGKDD.*
4. De Raedt, L., Guns, T. and Nijssen, S. (2010) Constraint Programming for Data Mining and Machine Learning. *In Proc. of AAAI.*

Active Selection of Datasetoids for Meta-Learning

Ricardo B. C. Prudêncio and **Carlos Soares** and **Teresa B. Ludermir** [12]

Abstract. Several meta-learning approaches have been developed for the problem of algorithm selection. In this context, it is of central importance to collect a sufficient number of datasets to be used as meta-examples in order to provide reliable results. Recently, some proposals to generate datasets have addressed this issue with successful results. These proposals include datasetoids, which is a simple manipulation method to obtain new datasets from existing ones. However, the increase in the number of datasets raises another issue: in order to generate meta-examples for training, it is necessary to estimate the performance of the algorithms on the datasets. This typically requires running all candidate algorithms on all datasets, which is computationally very expensive. One approach to address this problem is the use of active learning, termed active meta-learning. In this paper we investigate the combined use of active meta-learning and datasetoids. Our results show that it is possible to significantly reduce the computational cost of generating meta-examples not only without loss of meta-learning accuracy but with potential gains. Additionally, the results provide further evidence that datasetoids contain useful information for meta-learning.

1 Introduction

A large number of learning algorithms are available for data analysis nowadays. For instance, decision trees, neural networks, linear discriminants, support vector machines, among others, can be used in classification problems. After narrowing down the list of candidate algorithms taking into account problem-specific constraints (e.g., interpretability of the model), the goal of data analysts is to select the algorithm with higher chances to obtain the best performance on the problem at hand.

The above problem can be addressed by *meta-learning*, which treats it as a supervised machine learning task [3, 14]. A learning algorithm is used to model the relation between the characteristics of learning problems (e.g., application domain, number of examples, proportion of symbolic attributes) and the relative performance of a set of algorithms. This approach is known as meta-learning because it consists of learning about the performance of learning algorithms. The (meta-)data used for knowledge acquisition consists of a set of *meta-examples* representing characteristics of the learning problems and performance estimates of the candidate algorithms on those problems.

An important issue in the development of meta-learning systems for algorithm recommendation is the computational cost of generating the meta-data [3]. This implies running the candidate algorithms on all the training datasets. As the number of datasets used in most studies is limited (typically less than 100), this aspect has not been a very serious problem so far and, thus, has not received much attention, with the exception of [10]. Recently, several approaches to increase the number of data available for meta-learning have been proposed [6, 2, 15]. They enable the generation of meta-data with many hundreds or even thousands of meta-examples, thus making the problem of the computational cost of collecting the data for meta-learning a much more relevant one.

In this paper, we combine an active learning approach with a dataset generation method, datasetoids [15], to guide the meta-data collection process. This combination was tested in our work on an algorithm selection task. The contribution of this work is to show that these two methods simultaneously and successfully address two of the major issues in meta-learning: obtaining sufficient datasets for reliable meta-learning and reducing the computational cost of collecting meta-data.

We start by describing background information on meta-learning, including datasetoids, (Section 2) and active learning (Section 3). Then we describe how we can use active learning for datasetoid selection (Section 4). Next, we present the experimental setup used to evaluate the approach empirically (Section 5). We close the paper with conclusions and some ideas for future work (Section 6).

2 Meta-learning for Algorithm Selection

The meta-learning approach to algorithm selection is summarized in Figure 1. A database is created with meta-data descriptions of a set of datasets. These meta-data contain estimates of the performance of a set of candidate algorithms on those datasets as well as some meta-features describing their characteristics (e.g., number of examples in the dataset, entropy of the class attribute, mean correlation between the numerical attributes). A machine learning algorithm (the so-called *meta-learner*) is applied to this database to induce a model that relates the value of the meta-features to the performance of the candidate algorithms (e.g., the best algorithm on the datasets). For more information on meta-learning for algorithm recommendation, the reader is referred to [3, 14, 17] and references therein.

According to [3], three of the major issues in developing meta-learning systems are: (1) defining suitable meta-features, (2) gathering a sufficient number of meta-examples to obtain reliable meta-learning models and (3) reducing the computational cost of collecting the meta-data, which requires running the candidate algorithms on the training datasets. A significant amount of work has been devoted to designing meta-features (e.g., including general, statistical and information-theoretic measures, landmarkers and model-based measures).

The second issue is related to the availability of a number of datasets that is sufficient to enable reliable (meta-)induction. The UCI Repository [1] is the most common source of examples for

[1] Ricardo Prudêncio and Teresa Ludermir, Center of Informatics, Federal University of Pernambuco, Recife, Brazil, email:{rbcp,tbl}@cin.ufpe.br
[2] Carlos Soares, Faculdade de Economia, Universidade do Porto, Porto, Portugal, email:csoares@fep.up.pt

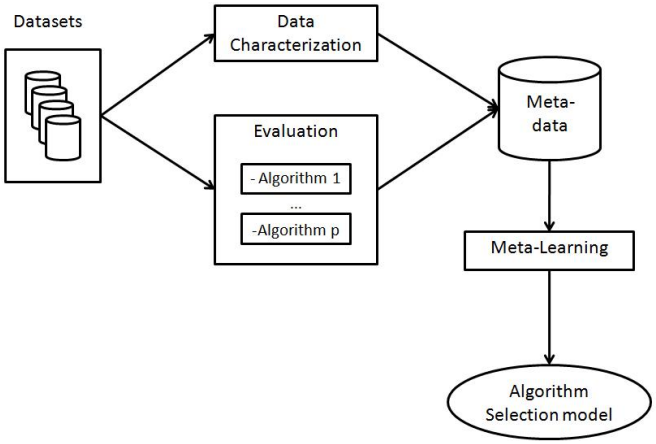


Figure 1. Meta-learning process for algorithm selection (adapted from [3])

meta-learning, however it contains slightly over 100 datasets. Given that each dataset represents one meta-example, most meta-learning research is based on approximately 100 meta-examples. This is a small number to ensure that highly reliable models are obtained, particularly in such a complex application such as meta-learning. This problem is receiving an increasing amount of attention recently. Two common approaches are the generation of synthetic datasets and the manipulation of existing ones [6, 2, 8].

In this work we use *datasetoids*, a very simple data manipulation approach to generate new datasets which was recently proposed [15]. A datasetoid is generated from a given dataset by switching the target attribute with an independent attribute (Figure 2). Thus, the target attribute of the dataset becomes an independent attribute in the datasetoid and the independent attribute selected in the dataset becomes the target attribute in the datasetoid. To generate datasetoids for classification, the process is repeated for every symbolic attribute of the dataset, thus creating as many datasetoids as there are symbolic attributes in the dataset. For instance, in [15], a number of 983 classification datasetoids was produced from a set of 64 UCI datasets. The newly generated datasets are called datasetoids because, although they are sets of data, they probably do not represent a learning application which is interesting in the corresponding domain (e.g., predicting which product was purchased by a customer, given the quantity of that product that was purchased and other information). Experiments on the problem of deciding whether to prune a decision tree using training meta-data containing datasetoids obtained significant improvements when compared to training meta-data that only contained datasets [15].

We highlight here that in order to collect performance meta-data it is necessary to execute the algorithms on the datasets, which is computationally expensive. This leads us to the third issue mentioned earlier, the need to reduce the computational cost of collecting the meta-data. If the number of datasets used for meta-learning is small, this is not very important. However, with current approaches to generate new datasets for meta-learning, such as datasetoids, its importance is increasing. To the best of our knowledge, the only attempt to address this problem is the use of active learning [10]. In the next section we describe active learning and its use to reduce the computational effort in meta-learning.

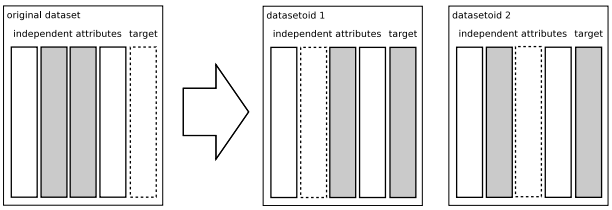


Figure 2. Illustration of the generation of two classification datasetoids from a dataset with two symbolic attributes (reproduced from [15])

3 Active Learning and Meta-learning

Active learning is a paradigm of Machine Learning in which the learning algorithm has some control over the examples on which it trains [5]. It has been used in many learning tasks to reduce the number of training examples, while maintaining (or in many cases improving) the performance of the learning algorithms (e.g., [7, 11, 12, 13]). Active learning is ideal for learning domains in which the acquisition of labeled examples is a costly process. In this case, active learning methods incrementally select unlabeled meta-examples to be labeled.

As discussed in the previous section, the cost of acquiring labels for meta-learning is quite computationally expensive, as it is necessary to execute the candidate algorithms on the datasets used for training. This makes meta-learning a good candidate problem for active learning techniques. In [10], the authors proposed the use of active learning to improve the generation of meta-examples. This proposal, termed as *Active Meta-learning*, is illustrated in Figure 3.

Differently from the conventional meta-learning introduced in the previous section, the evaluation of the candidate algorithms is performed only in a selected subset of the available datasets. For this, an active learning module receives as input a set of *unlabeled* meta-examples, i.e., datasets in which the candidate algorithms were not yet run. Hence, the class labels of these meta-examples are not known. The selection of unlabeled meta-examples is performed based on a pre-defined criterion which takes into account the meta-features of the problems and the current knowledge of the meta-learner (i.e., the meta-examples already labeled). The candidate algorithms are then evaluated on the selected problems and the best algorithm on each of them becomes the label of the corresponding meta-example. Then, the meta-learning module acquires knowledge from the database of labeled meta-examples generated by the active learning module. The acquired knowledge may be refined as more labeled meta-examples are provided.

The Active Meta-learning approach was empirically evaluated in [10]. The active learning algorithm used was an *uncertainty sampling method* originally proposed in [7] and based on the k-NN algorithm. This method selects unlabeled examples for which the current k-NN learner has high uncertainty in its prediction. The uncertainty of k-NN was defined in [7] as the ratio of: (1) the distance between the unlabeled example and its nearest labeled neighbor; and (2) the sum of the distances between the unlabeled example and its nearest labeled neighbor of every class. A high value of uncertainty indicates that the unlabeled example has nearest neighbors with similar distances but conflicting labeling. Hence, once an uncertain unlabeled example is labeled, it is expected that the uncertainty in its neighborhood is reduced.

In the context of meta-learning, let E be the set of labeled meta-

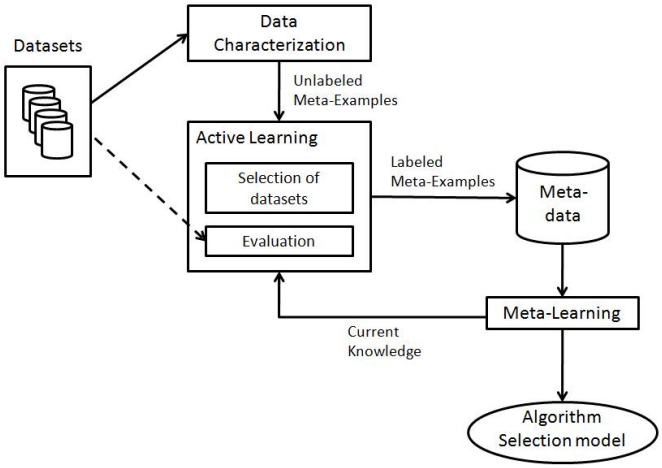


Figure 3. Active meta-learning process

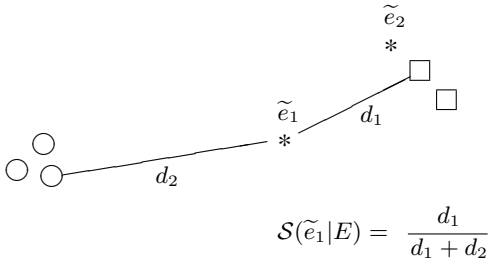


Figure 4. Illustration of Uncertainty Sampling.

examples. Let $\mathcal{C} = \{c_1, \ldots, c_L\}$ be the domain of the class attribute C, with L possible class labels, representing the set of candidate algorithms. Each labeled meta-example e_i is represented as the pair $(\mathbf{x}_i, C(e_i))$ storing: (1) the description $\mathbf{x}_i$ of the problem e_i, where $\mathbf{x}_i = (x_i^1, \ldots, x_i^m)$ is a vector of m meta-features; and (2) the class attribute C associated to e_i, i.e., $C(e_i) = c_l$, where $c_l \in \mathcal{C}$.

Let $\widetilde{E}$ be the set of unlabeled meta-examples. Let E_l be the subset of labeled meta-examples associated to the class label c_l, i.e., $E_l = \{e_i \in E | C(e_i) = c_l\}$. Given E, the classification uncertainty of k-NN for each $\widetilde{e} \in \widetilde{E}$ is defined as:

$$\mathcal{S}(\widetilde{e}|E) = \frac{\min_{e_i \in E} dist(\widetilde{\mathbf{x}}, \mathbf{x}_i)}{\sum_{l=1}^{L} \min_{e_i \in E_l} dist(\widetilde{\mathbf{x}}, \mathbf{x}_i)} \quad (1)$$

where $\widetilde{\mathbf{x}}$ is the description of the unlabeled meta-example $\widetilde{e}$ and $dist$ is the distance function adopted by the k-NN algorithm. The unlabeled meta-examples in $\widetilde{E}$ are selected according to the following probabilities:

$$p(\widetilde{e}) = \frac{\mathcal{S}(\widetilde{e}|E)}{\sum_{\widetilde{e}_i \in \widetilde{E}} \mathcal{S}(\widetilde{e}_i|E)} \quad (2)$$

The above probability is just a normalized value of the uncertainty. The roulette-wheel algorithm is often used to sample the unlabeled meta-examples according to their associated probabilities. In this method, probability of a meta-example being sampled is proportional to its uncertainty. The sampled unlabeled meta-example is labeled (i.e., the class value $C(\widetilde{e})$ is defined) by estimating the performance of the candidate algorithms on the corresponding dataset. The new labeled meta-example $(\widetilde{\mathbf{x}}, C(\widetilde{e}))$ is then included in the meta-data.

Figure 4 illustrates the use of the uncertainty sampling method. Circles and squares represent two different classes of labeled examples. The stars named as $\widetilde{e}_1$ and $\widetilde{e}_2$ represent two unlabeled examples which are candidates to be labeled. The example $\widetilde{e}_2$ would be less relevant since it is very close to the labeled examples of one specific class. By deploying the uncertainty sampling method, the $\widetilde{e}_1$ would have a higher probability to be sampled since it is more equally distant from labeled examples of different classes.

We highlight that the experiments on Active Meta-learning reported in [10] were carried out on a relatively small number of datasets . Thus, although those experiments revealed promising results, they are not suitable to assess the potential benefits of Active Meta-Learning accurately.

4 Active Selection of Datasetoids

The approach to generate datasetoids described earlier is an interesting solution to augment the number of training examples for meta-learning and hence, to improve its performance in algorithm selection tasks, as shown in [15]. A potentially large number of datasetoids can be produced even by applying simple operators for problem manipulation. Producing a set of meta-examples from *all* datasetoids however can be costly since for each available datasetoids it is necessary to execute all candidate algorithms, to estimate their performance and, thus, label the corresponding meta-example.

Additionally, not all datasetoids are necessarily useful because they may contain information that is irrelevant or redundant for meta-learning. In [15], the author discussed a number of potential anomalies in datasetoids which can lead to useless meta-examples. For instance, datasetoids are random when the corresponding target attribute is completely unrelated to all the other attributes. Also, redundant datasetoids can be produced when two or more attributes of a dataset represent exactly the same property or when one attribute is completely determined by another one (e.g., the zip code and the city, where the former determines the latter). Hence, it would be more effective to consider only the most relevant datasetoids for the generation of meta-examples.

Motivated by the above considerations, in the current work we combine the use of Active Meta-Learning and datasetoids. Figure 5 illustrates our proposal, which is similar to the Active Meta-Learning presented in Figure 3. However, it expands the training set with datasetoids. It employs a dataset manipulation operator to generate datasetoids and augment the number of candidate datasets for meta-example generation. Initially, a number of datasets related to real problems is collected (for instance, from data repositories such as the UCI [1]). By applying specific operators for problem manipulation, a set of datasetoids is produced by modifying the original datasets. Data characterization is then applied to both the datasets and datasetoids in order to produce a large set of unlabeled meta-examples. The selection of unlabeled meta-examples is done using the uncertainty sampling method described in the previous section.

Compared to the straightforward use of datasetoids as a mechanism for meta-data generation [15], this work incorporates an active

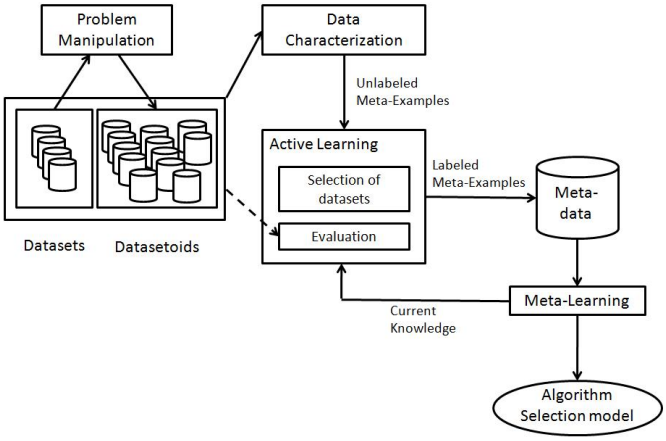


Figure 5. Active meta-learning process combined with datasetoids

learning mechanism to select only the most relevant datasetoids, and thus, reduce the computational cost. In turn, compared to the original Active Meta-Learning work, we augment the number of candidate problems for meta-example generation by producing datasetoids from real datasets, thus creating a more suitable context to assess its usefulness. This means that we simultaneously address two of the most important issues in meta-learning: obtaining sufficient datasets for reliable meta-learning and reducing the computational cost of collecting meta-data.

In the current work, we adopted the uncertainty sampling approach for active learning described in the previous section. We highlight that this approach has some drawbacks reported in the literature. First, it is sensible to the presence of outliers, which may be considered as uncertain examples but should not be included in the training set [9]. Also, it selects uncertain examples without considering whether they belong to dense regions in the instance space. Hence, it may choose isolated examples that will not be relevant to significantly affect the current classifier [7]. Despite these drawbacks, the uncertainty sampling method is simple and achieved positive results in [10]. Additionally, it is based on the k-NN algorithm which has previously obtained positive results in meta-learning problems with few meta-examples [4].

5 Experiments and Results

We empirically evaluate the proposed approach on the same meta-learning task used in [15]. It consists of predicting, a priori, if pruning a decision tree will improve the quality of the model or not. The implementation of the algorithm for induction of decision trees used is from the `rpart` package of the R statistical package [16]. Pruned decision trees were obtained using the default parameters of this implementation. Unpruned trees were obtained by setting the complexity parameter (`cp`) to 0, which means that any split will be accepted as long as it brings some improvement to the model fit (> 0), no matter how small it is. The measure of performance used was classification accuracy, estimated using 10-fold cross-validation. The class label of each meta-example is based on these results. The possible values are `p`, `u` or `t`, meaning, respectively, the winner is the pruned tree, the unpruned tree or that they are tied.

Concerning the meta-features to characterize the datasets (and the datasetoids), we adopted the ones that were used in [15]. They are (1) the entropy of the class and (2) the average entropy of the attributes [4]. These meta-features are expected to contain some information about the behavior of decision trees because this algorithm uses the concept of entropy. But, most importantly, although there are certainly other meta-features that could contribute to improve the meta-learning results, the use of measures that previously obtained good results enables us to focus on the main goal of this paper, which is to test the combination of active meta-learning and datasetoids.

The set of problems used to generate meta-examples are also the same 64 UCI datasets from [15] and 983 corresponding datasetoids. Table 1 presents the class distribution, both in the meta-data obtained from the datasets as in the meta-data obtained from the datasetoids. The table shows that the class distributions are not exactly the same, which indicates that there may be some underlying differences between datasetoids and datasets. However, we also note that the differences are not so large and that the relative proportions are the same (i.e., ties are the most common, followed by the case when pruning improves accuracy).

Table 1. Class distribution (%) of the meta-data corresponding to the datasets and to the datasetoids

metadata	pruned tree wins (`p`)	unpruned tree wins (`u`)	tie (`t`)
datasets	36	23	41
datasetoids	37	10	53

In the following sub-sections, we present the experiments performed in our work. First, we evaluate the value of using datasetoids for the k-NN meta-learner. Following, we present the experiments performed to verify the viability of combining active meta-learning and datasetoids.

5.1 k-NN

In the original datasetoids paper [15], the k-NN algorithm was not used as a meta-learner. So, we start by testing whether similar gains can also be obtained by using datasetoids as training data for the k-NN. In this experiment, the k-NN algorithm was executed with the normalized euclidean distance and $k = 1$.

Given that the datasetoids are generated from datasets, they cannot be treated as independent problems. Therefore, to estimate meta-learning performance, we adopted the same methodology as in [15]. Firstly, predicting which algorithm is the best on the datasetoids is not relevant. As stated above, datasetoids are not interesting as applications *per se*. Therefore, only the original UCI datasets are used as test set. Additionally, to ensure independence between the training and test sets, we must guarantee that the datasetoids generated from a given dataset are not included in the training set used to make a prediction for that dataset. As we use a leave-one-out (LOO) approach, the meta-learner is evaluated on one dataset at a time. Thus, the meta-examples corresponding to the datasetoids obtained from that dataset are removed from the training meta-data used to make a prediction for that dataset. The measure of meta-learning performance is the classification accuracy, i.e., the proportion of datasets for which a correct decision was made, in terms of whether it is best to prune or not, or if there is a tie.

We compare the results obtained using datasetoids as training data to the results of using only the datasets. In the latter case, LOO consists of evaluating the meta-learner for each dataset using all the other

datasets for training. Additionally, we have tested a combination of datasets and datasetoids. In this case, the training data for a given test dataset includes all the other datasets plus the datasetoids that were not generated from the test dataset, as previously explained.

Table 2 shows the accuracy obtained by the k-NN algorithm in all cases. We also include the results of the algorithms adopted in [15] since the experimental setup is exactly the same: decision trees (`dt`), linear discriminant (`ld`), random forest (`rf`), support vector machines (`svm`) and multi-layer perceptron (`nn`) as learning algorithms. Concerning k-NN, the best accuracy was obtained when all meta-data (datasets and datasetoids) were considered, followed by the use of datasetoids alone. Statistical gain in accuracy was verified when all meta-data were used by k-NN (applying a paired t-test at a 95% level of confidence). These results additionally support the claim that datasetoids contain useful information for meta-learning [15]. Comparing the results of k-NN with the other algorithms, we can make two interesting observations. Firstly, the performance of k-NN using only the datasets is better than the performance of the other meta-learning algorithms. Secondly, when datasetoids are used for training, the advantage of k-NN over the other algorithms is not so clear. This supports previous claims that k-NN is a suitable meta-learning algorithm when the number of available datasets is small [4].

Table 2. Results of using datasetoids on the meta-learning task. The numbers represent accuracy at the meta-level (%). The row "default" represents the proportion of the most frequent class in the meta-data of the datasets. All results except the ones for k-NN are reproduced from [15].

algorithm/training set	datasets	datasetoids	datasets+ datasetoids
`k-NN`	52	55	58
`dt`	41	55	55
`ld`	41	47	48
`rf`	47	62	59
`svm`	41	52	53
`nn`	45	53	50
default	41		

5.2 Active Meta-learning

In a second battery of experiments, we evaluated the combined application of the active meta-learning and datasetoids, as described in Section 4. Active meta-learning is done using the uncertainty sampling method described in Section 3.

As in the previous experiment, the original 64 datasets were used as test set. At each step of the LOO procedure, one dataset is left out and the uncertainty sampling method is iteratively used to include labeled meta-examples in the training meta-data. In these experiments we start with an empty set of problems and allow the method to sample up to 500 training meta-examples (about 50% of the available candidate problems). We highlight that, for the reasons mentioned in Section 5.1, we exclude from the training set the datasetoids generated from the test dataset. At each included meta-example, the meta-learner is judged on the test dataset left out, receiving either 1 or 0 for success or failure. Hence, a curve with 500 binary judgments is produced for each test dataset. Finally, the curve of accuracy obtained by the meta-learner is computed by averaging the judgments over the 64 steps of the leave-one-out experiment.

As a basis for comparison, we tested the Random Sampling method for selecting unlabeled problems as an alternative to uncertainty sampling. According to [7], despite its simplicity, the random method has the advantage of performing a uniform exploration of the example space. The comparison with this method enables us to assess the gains obtained by using uncertainty sampling as meta-example selection method. Given that it is possible that the results of both sampling methods are affected by the first meta-example selected, we repeat the experiments 100 times.

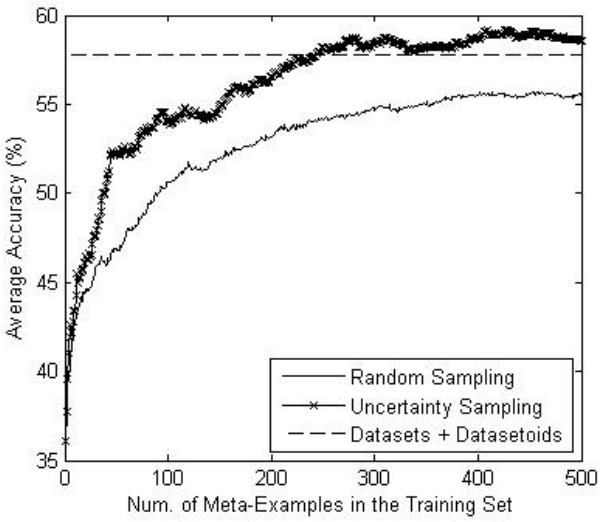


Figure 6. Average accuracy obtained by uncertainty and random sampling varying the number of meta-examples in the training set. The dashed line represents the accuracy obtained when all available datasets and datasetoids are used to make predictions.

Figure 6 presents the average curves of accuracy obtained by uncertainty sampling and the random method. As it can be seen, for both methods the accuracy of the k-NN meta-learner increases as the number of meta-example increases. However, the accuracy rates obtained by deploying the uncertainty sampling method were, in general, higher than the accuracy rates obtained by random sampling. The same level of accuracy obtained using all datasets and datasetoids (58%) was achieved when 245 meta-examples were included in the training set, representing less than 30% of the available problems. This result is similar to the best accuracies obtained in [15], however in our results a lower number of meta-examples was required. Furthermore, the Active Meta-learning enables us to take advantage of datasets even with stronger computational limitations. The figure shows that with as little 44 meta-examples, it is possible to obtain good results, with accuracies greater than 52%. This number of meta-examples represents less than 5% of the candidate problems available in average at each LOO step.

We also highlight that the standard deviation values observed by using the uncertainty sampling method were lower compared to the standard deviation values of random sampling (see Figure 7). This indicates that, not only it is more accurate than random sampling, it is also more robust.

6 Conclusion

We proposed the combination of Active Meta-Learning and datasetoids to simultaneously address two of the most important issues of

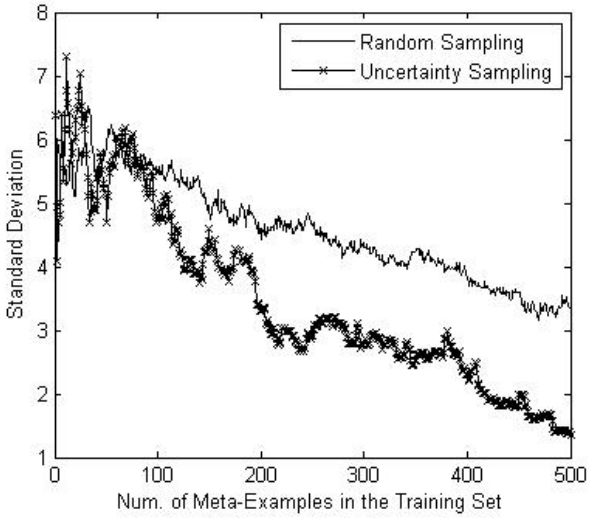


Figure 7. Standard deviation of accuracy obtained by uncertainty and random sampling varying the number of meta-examples in the training set.

meta-learning for algorithm recommendation: augmenting the number of datasets to produce meta-examples and reducing the computational cost of collecting meta-data.

Our results show that it is possible to take advantage of the large number of meta-examples provided by datasetoids methods without incurring significant extra computational costs. Additionally, if sufficient resources are available, it is even possible to achieve improvements, possibly due to the elimination of irrelevant and misleading meta-examples.

These results were obtained with a simple active learning method with known limitations (e.g., sensitivity to outliers). This opens up a number of exciting possibilities for improvement, by using more complex active learning methods, possibly adapting them for meta-learning and also other meta-learning algorithms.

We point out that our experiments were restricted to a single meta-learning problem. Although this is still common practice in the field, we will test the approach on other meta-learning problems, with different learning problems (e.g., regression), sets of base-level algorithms and meta-features.

Finally, an important issue which was not taken into account in our work is the cost of computing the meta-attributes. In fact, producing unlabeled meta-examples from *all* datasetoids may not be adequate when the extraction of meta-features is computationally expensive. Hence, in future work, we intend to propose other active learning strategies which also consider the cost of the meta-attributes.

Acknowledgements. The authors would like to thank CNPq, CAPES and FACEPE (Brazilian Agencies) for their financial support. This work was also partially funded by FCT (Programa de Financiamento Plurianual de Unidades de I&D and project Rank! - PTDC/EIA/81178/2006)

REFERENCES

[1] A. Asuncion and D.J. Newman. UCI machine learning repository, 2007.

[2] Hendrik Blockeel and Joaquin Vanschoren, 'Experiment databases: Towards an improved experimental methodology in machine learning', in *Knowledge Discovery in Databases: PKDD 2007, 11th European Conference on Principles and Practice of Knowledge Discovery in Databases, Proceedings*, volume 4702 of *Lecture Notes in Computer Science*, pp. 6–17. Springer, (2007).

[3] P. Brazdil, C. Giraud-Carrier, C. Soares, and R. Vilalta, *Metalearning: Applications to Data Mining*, Cognitive Technologies, Springer, 2009.

[4] P. Brazdil, C. Soares, and J.P. da Costa, 'Ranking learning algorithms: Using IBL and meta-learning on accuracy and time results', *Machine Learning*, **50**(3), 251–277, (2003).

[5] D. Cohn, L. Atlas, and R. Ladner, 'Improving generalization with active learning', *Machine Learning*, **15**, 201–221, (1994).

[6] M. Hilario and A. Kalousis, 'Quantifying the resilience of inductive classification algorithms', in *Proceedings of the Fourth European Conference on Principles of Data Mining and Knowledge Discovery*, eds., D.A. Zighed, J. Komorowski, and J Zytkow, pp. 106–115. Springer-Verlag, (2000).

[7] M. Lindenbaum, S. Markovitch, and D. Rusakov, 'Selective sampling for nearest neighbor classifiers', *Machine Learning*, **54**, 125–152, (2004).

[8] N. Macià, A. Orriols-Puig, and E. Bernadó-Mansilla, 'Genetic-based synthetic data sets for the analysis of classifiers behavior', in *Proceedings of 15th International Conference on Hybrid Intelligent Systems*, pp. 507–512, (2008).

[9] I. Muslea, S. Minton, and C. Knobrock, 'Active learning with multiple views', *Journal of Artif. Intel. Research*, **27**, 203–233, (2006).

[10] R. B. C. Prudêncio and T. B. Ludermir, 'Selective generation of training examples in active meta-learning', *International Journal of Hybrid Intelligent Systems*, **5**, 59–70, (2008).

[11] H. Raghavan, O. Madani, and R. Jones, 'Active learning with feedback on both features and instances', *Pattern Recognition Letters*, **7**, 1655–1686, (2006).

[12] G. Riccardi and D. Hakkani-Tur, 'Active learning - theory and applications to automatic speech recognition', *IEEE Transactions on Speech and Audio Processing*, **13**(4), 504–511, (2005).

[13] I. Sampaio, G. Ramalho, V. Corruble, and R. Prudêncio, 'Acquiring the preferences of new users in recommender systems - the role of item controversy', in *Proceedings of the ECAI 2006 Workshop on Recommender Systems*, pp. 107–110, (2006).

[14] K. Smith-Miles, 'Cross-disciplinary perspectives on meta-learning for algorithm selection', *ACM Computing Surveys*, **41**(1), 1–25, (2008).

[15] C. Soares, 'Uci++, improved support for algorithm selection using datasetoids', *Lecture Notes in Computer Science*, **5476**, 499–506, (2009).

[16] R Development Core Team. R - a language and environment for statistical computing, 2008.

[17] R. Vilalta and Y. Drissi, 'A perspective view and survey of meta-learning', *Journal of Artificial Intelligence Review*, **18**(2), 77–95, (2002).

Predicting Classifier Performance using Data Set Descriptors and Data Mining Ontology

Derry Tanti Wijaya[1], **Alexandros Kalousis**[1], **and Melanie Hilario**[1]

Abstract. One purpose of meta-learning is to reduce the set of candidate algorithms that users need to experiment with before they select the algorithm that performs the best on their data set. In this paper, we present a preliminary study that combines both insights from data set characteristics (computed using data set descriptors on the data) and algorithm characteristics and similarities (computed using similarity kernels on data mining ontology) to reduce the set of candidate algorithms which users need to experiment with, by giving an a priori prediction of algorithm performance on the data set. We base our prediction of algorithm performance on the assumption that *algorithms with similar characteristics will have similar performance on similar data sets* (data sets with similar characteristics). We evaluate our proposed method for the task of classification and observe an increase in prediction accuracy when we use both data set characteristics and algorithm characteristics to predict classifier performance compared to when we use data set characteristics or algorithm characteristics alone. Another advantage of our proposed method is its use of a data mining ontology to organize algorithm characteristics and to compute similarities between two algorithms. The advantage of using a data mining ontology is that we can even predict the performance of an algorithm we have never seen before in previous experiments (hence whose performance has never been observed), as long as the algorithm characteristics and similarities to other algorithms that we have used (whose performance have been observed) can be computed from the ontology. If this information can be computed, we can predict the new algorithm's performance on a data set by relating it to the observed performance of other algorithms – algorithms that possess similar algorithm characteristics – on similar datasets.

1 INTRODUCTION

Due to the myriad of classification algorithms available, users faced with a classification task may be required to experiment with a broad range of classification algorithms and parameter settings before they can select the classifier that performs best on their data set according to some performance criterion. In this case, insights that can help users narrow down the set of candidate algorithms will be beneficial.

Unfortunately such insights cannot be gained just from theoretical studies of classifier characteristic (e.g. performance bounds) on all possible data sets because the empirical performance of classifiers is also known to be critically dependent on the characteristics of the data set being investigated [1]. Algorithm characteristics cannot stand alone; they must be tied to data characteristics in predicting performance.

Nor can such insights be gained just from formulating and correlating descriptors of data set characteristics with the observed performance of the algorithms treated as black boxes [1, 2] since there is also a close link between data set descriptors and algorithms (i.e. some data set descriptors can explain well the performance of some algorithms but not others [3]). In treating the algorithms as black boxes, little or no analysis can be done on the reason why certain data set descriptors are good at predicting certain algorithm's performance. Since algorithms are treated as 'black boxes', given an algorithm that has never been used before in previous experiments, it will be difficult to predict its performance because its characteristics and relationships to the 'black boxes' are unknown. Data set characteristics cannot stand alone; they must be tied to the algorithm in predicting performance.

Hence, there is a need to combine information from data set characteristics and algorithm characteristics to gain insights into the applicability of an algorithm to a data set at hand. It is the interplay, the combination of both data set characteristics and algorithm characteristics that affects performance (good or bad) of an algorithm on a data set.

In the e-LICO project [4], we have started to pry open the black boxes of algorithms to sort out salient algorithm characteristics and present them in the form of a data mining ontology [5]. In this paper, we present a preliminary study of using both data set characteristics (computed using data set descriptors [2] on the data set) and algorithm characteristics (computed using similarity kernels [6] on our data mining ontology [5]) as features to predict the performance of algorithms.

We evaluate our proposed method for the task of classification using previously executed experiments as our training and testing data. We report an increase in the accuracy of prediction when compared to using data set descriptors alone or algorithm characteristics alone.

Another advantage of our method is its ability to predict the performance of a classifier that has never been used in previous experiments as long as its characteristics and similarities to other classifiers used in previous experiments can be computed from the ontology.

This paper is organized as follows. First in Section 2, we discuss related meta-learning work in predicting the applicability of a classifier on a data set. Next, we describe our proposed method in Section 3. In Section 4, we describe our experimental setup. We present and discuss the results of our experiments in

[1] Laboratoire d'Intelligence Artificielle, Centre Universitaire d'Informatique, Université de Genève, 7 Route de Drize, 1227 Carouge, Switzerland, Email: {Derry.Wijaya, Alexandros.Kalousis, Melanie.Hilario}@unige.ch.

Section 5. Section 7 presents our conclusions and provides directions for future work.

2 RELATED WORK

Since there are a large number of algorithms that have been discovered for the task of classification, many studies have been conducted to analyze the different classifier behaviours on different data sets.

Typical empirical studies usually start by experimenting with a range of classifiers and their parameter settings on a specific class of problems and conclude with a presentation of performance of the different classifiers with little or no analysis behind the classifier's success or failure. Theoretical studies on the other hand, attempt to analyze classifier behaviour and its performance bounds for all possible problems resulting in typically weak performance bounds [2].

Since empirically observed behaviour of classifiers has been known to be strongly linked to the data set being investigated, a lot of research has been conducted to understand and measure such classifier and data set dependency and use this understanding to explain classifier behaviour on the data set.

To do this, one approach is to understand this 'classifier and data set dependency' from the point of view of the data set. Such data-centric meta-learning typically pays attention to the good characterization of the data set [1, 2, and 3]. The studies conduct comparative studies of several classifiers and relating their performances to a range of proposed data set characteristics descriptors. The studies are conducted on either real or synthetic data sets, but the focus is always on finding a good set of data set descriptors – measures of data set characteristics that can predict classifier performance well – with classifiers themselves treated as 'black boxes'. Data set characteristics measures used are often statistical or information theoretical in nature like in the Statlog project [1] while some are geometrical in nature, focusing on describing the complexity of the class boundary [2, 3]. Unfortunately with such an approach, there is inevitably an endless supply of data set descriptors that can be used (since there can be many ways of describing a data set!). Hence, the reliability of predictions is often questionable: yet another descriptor can always be found to explain to (yet) another degree the performance of the classifiers. The merit of each descriptor and why some descriptors are only good for predicting the performance of some classifiers [3] are unknown since the classifiers themselves are treated as 'black boxes' and data set descriptors are defined independently of the classifiers being investigated. It is also difficult to extend the result of these studies to other classifiers whose performance has never been observed before because there is no way of relating these new classifiers to the 'black boxes' of already observed classifiers.

Another line of approach to understanding this 'classifier and data set dependency' is to use classifiers to chart the space of data sets. An example is the so-called landmarking approach which tries to relate the performance of some simple learners – the landmarkers – to the performance of some other algorithm [7]. The idea is to apply the landmarkers to a specific problem (data set) at hand and use the performance of the landmarkers as meta-data for learning the performance of some other algorithm on the data set. By doing this, the landmarkers indirectly characterize a data set by the performance of the landmarkers on that data set. Although landmarking comes close to describing a data set from the point of view of algorithms, just like the choice of suitable data set descriptors, the choice of suitable landmarkers is potentially endless. Unfortunately, the landmarking approach can only be successful when landmarkers chosen are able to measure properties of a data set that are relevant to the success or failure of some learning algorithm on that data set [7]. Landmarking therefore cannot stand alone with data set and algorithm treated as 'black boxes'. It must be tied to the salient characteristics of the data set and the characteristics of the algorithm whose performance on the data set we are trying to predict.

Hence, we believe that to be able to truly understand classifier and data set dependency, we need to understand not only the data set but also the classifier (the black boxes of both data set and algorithm must be pried open) and use both to explain classifier behaviour on the data set.

3 PROPOSED METHOD

In this paper we define an experiment e as a tuple consisting of three elements: data set (D), classification algorithm (A), and performance (P):

$$e_i = (D(e_i), A(e_i), P(e_i))$$

where $D(e_i)$ represents the data set used in experiment e_i; $A(e_i)$ represents the classification algorithm and its parameter settings used in experiment e_i; and $P(e_i)$ represents the binary labeling of the performance (i.e. 'best' or 'rest') of the algorithm $A(e_i)$ on the data set $D(e_i)$. 'Best' performance label means the algorithm performs the best on the data set according to some performance criteria, while 'rest' performance label means the algorithm does not perform the best on the data set (i.e. there are other algorithms that perform better on the data set).

We then define distance (or inverse similarity) between two experiments e_i and e_j as the weighted sum of the distance between their data sets and the distance between their algorithms:

$$dist(e_i, e_j) = \alpha * dist_{DCT}(D(e_i), D(e_j)) + \beta * dist_{ONT}(A(e_i), A(e_j))$$

where $dist(e_i, e_j)$ represents distance between experiments e_i and e_j; $dist_{DCT}(D(e_i), D(e_j))$ represents the Euclidian distance between the two vectors that respectively contain the values of data set descriptors of $D(e_i)$ and $D(e_j)$; and $dist_{ONT}(A(e_i), A(e_j))$ represents the distance between $A(e_i)$ and $A(e_j)$ as measured in the ontology. α and β are two real-valued constants between 0 and 1 (and $\alpha + \beta = 1$) that represent the weights (of importance) assigned to the two component distances: $\alpha > \beta$ means greater emphasis is placed on the distance between data sets used in the experiments while $\alpha < \beta$ means greater emphasis is placed on the distance between algorithms used in the experiments.

From these definitions and a database of previous (executed) experiments as the training data, given a new (unexecuted) experiment e_z as a test instance (i.e. a tuple consisting of a data set $D(e_z)$, an algorithm $A(e_z)$, and an unknown performance label $P(e_z)$), we can compute e_z distances to all the previous (executed) experiments and find previous experiments that are nearest (having smallest distance) to it. Using a method not unlike the k-nearest neighbor classification algorithm [8], we can predict (classify) the unknown performance $P(e_z)$ based on the majority

performance of its k-nearest experiments neighbors. $P(e_z)$ is classified as 'best' if the majority of its nearest neighbors has 'best' performance label and classified as 'rest' otherwise.

Since the distance between two algorithms is measured on the data mining ontology, an advantage of our proposed method is that we can classify the performance of an algorithm we have never used before in our training data as long as the algorithm exists in the ontology: we can compute its distance to algorithms we have used before in previous experiments, find its nearest neighbor (the experiment that has the most similar algorithm and most similar data set), and predict the performance of the algorithm from the performance of its nearest neighbor.

4 EXPERIMENTAL SETUP

4.1 Experiment: Data Set, Algorithm, and Performance Label

To evaluate our proposed method, we first conduct experiments to populate our database of previous (executed) experiments. We conduct these experiments on a collection of 65 cancer data sets (a link to these data sets, their sources and descriptions can be found in [9]). This collection of data sets has characteristics common to biological data which is high dimensionality and few instances (the average number of features for this collection is 79 while the average number of instances is 15,269) Due to their high dimensionality and few instances, these data sets constitute a difficult classification task.

On each of these data sets, we execute these classification algorithms: *1-Nearest Neighbour*, *J48 Decision Tree*, *Simple Cart Decision Tree*, *Naïve Bayes*, *Random Tree*, *libSVM with Gaussian Kernel*, and *libSVM with Linear Kernel* (we use the implementations of these algorithms and their default parameter settings in RapidMiner software [10]). For each data set $D(e_i)$ and each algorithm $A(e_i)$ that constitute an experiment e_i, we note the average accuracy returned by a 10-fold cross validation in RapidMiner as the measure of the performance.

From these continuous accuracy scores, we need to assign a binary performance label $P(e_i)$: 'best' or 'rest' to each experiment e_i.

To do this, for each data set we identify the best performing algorithm based on accuracy. We then assign performance label 'best' to the experiment to which this best performing algorithm belongs. Then we use a Wilcoxon-test [11] with a 95% confidence level to identify which other algorithms are equivalent to (i.e. do not produce significantly different models on the data set from) the best performing algorithm. We also assign the performance label 'best' to experiments to which these algorithms (that are equivalent to the best performing algorithm) belong. Then, the rest of the experiments that do not have the best performing algorithms on the data set we label as 'rest'.

Using these procedures we obtain a set of experiments, each a tuple consisting of a data set, an algorithm, and a performance label. In total we have 455 experiments (= 65 data sets * 7 algorithms), 215 of which are labelled as 'best' and 240 are labelled as 'rest'.

The distribution of algorithm performance (i.e. the percentage of the 65 data sets on which each algorithm performs best) can be found in Table 1.

Algorithm	% Best
1NN	64.6
J48	58.5
Simple Cart	63.1
NB	63.1
Random Tree	27.7
SVM-Gaussian	16.9
SVM-Linear	36.9

Table 1. Distribution of Algorithm Performance on the 65 Cancer Data Sets

4.2 Data Set Descriptors

To measure distance between two data sets, we represent each data set as a vector of its characteristic values measured by a set of data set descriptors. Table 2 summarizes data set descriptors that we use in our experiment. These descriptors are formulated in [2] to measure the complexity of a data set from the geometrical perspective (the complexity of class boundary) which is particularly relevant to the task of classification.

1	Maximum Fisher's discriminant ratio
2	Volume of overlap region
3	Maximum (individual) feature efficiency
4	Minimized error by linear programming (LP)
5	Error rate of linear classifier by LP
6	Nonlinearity of linear classifier by LP
7	Fraction of points on boundary
8	Error rate of 1-nearest neighbor (NN) classifier
9	Nonlinearity of 1NN classifier
10	Ratio of average intra/inter class NN distance
11	Fraction of points with associated adherence subsets retained
12	Average number of points per dimension

Table 2. List of Data Set Descriptors Used

Now that each data set is represented by a vector of its characteristic values, we can measure distance between two data sets: $dist_{DCT}(D(e_i), D(e_j))$ as the Euclidian distance between the two representation vectors of $D(e_i)$ and $D(e_j)$.

4.3 Algorithm Distance in the Ontology

To measure distance between two algorithms (represented as concepts) in our data mining ontology [5], we use kernel methods for ontology proposed in [6]. Specifically, we use identity kernel, common class kernel, property kernel (both object property and data property), and path length distance.

Identity kernel simply performs a binary check on the identity of two concepts and return 1 if the two concepts are equal and 0 if the two concepts are not equal. Using identity kernel, distance between two algorithms in the ontology is measured as:

$$dist_{ONT\text{-}identity}(A(e_i), A(e_j)) = 1 - k_{identity}(A(e_i), A(e_j))$$

where $k_{identity}(A(e_i), A(e_j))$ is the identity kernel between concepts representing $A(e_i)$ and $A(e_j)$ in the ontology, as defined in [6].

Common class kernel on the other hand counts the number of parent and/or ancestor concepts that two concepts in the ontology have in common. We normalize this kernel to obtain only values from 0 to 1. Using common class kernel, distance between two algorithms in the ontology is measured as:

$$dist_{ONT\text{-}class}(A(e_i), A(e_j)) = 1 - k_{class}(A(e_i), A(e_j))$$

where $k_{class}(A(e_i), A(e_j))$ is the normalized count of the number of parent and/or ancestor concepts that the two concepts representing $A(e_i)$ and $A(e_j)$ in the ontology have in common.

Property kernel counts the number of object property and data property that the two concepts in the ontology have in common including their values (i.e. similar concept referenced by a similar object property or similar datatype referenced by a similar data property). It is basically a sum of the data property kernel and object property kernel in [6] without following on a concept referenced by an object property (i.e. we do not follow further dependencies from concepts referenced by object properties). Similar to common class kernel, we normalize this kernel to obtain only values from 0 to 1. Distance between two algorithms in the ontology is then measured as:

$$dist_{ONT\text{-}property}(A(e_i), A(e_j)) = 1 - k_{property}(A(e_i), A(e_j))$$

where $k_{property}(A(e_i), A(e_j))$ is the normalized count of the number of data and object properties (and their values) that the two concepts representing $A(e_i)$ and $A(e_j)$ in the ontology have in common.

Path length distance, as the name suggests, finds the path between two concepts in the ontology and measures its length. Using this measure, distance between two algorithms in the ontology $dist_{ONT\text{-}pathLength}(A(e_i), A(e_j))$ is measured simply as the normalized path length between the two concepts representing the algorithms $A(e_i)$ and $A(e_j)$ in the ontology.

We compute these distance measures on our data mining ontology which provides a conceptualization of data mining tasks, methods/algorithms and data sets [5].

Since the ontology is used in this paper for the purpose of computing distance between two algorithms, in Figure 1 we show a snapshot of the *Classification Algorithm* sub tree. In this snapshot, as we go down the sub tree, the classification algorithm split into more specialized algorithms which in turn give rise to formally specified algorithms such as those on the right side of the figure.

From this snapshot in Figure 1, we can also measure the path length between two algorithms. For example, using Figure 1 we can see that the path length between *CART-Algorithm* concept (for *Simple Cart* algorithm) and *C4.5-Algorithm* concept (for *J48* algorithm) is 2, while the path length between *CART-Algorithm* and *Naïve Bayes Normal Algorithm* is 5.

4.4 Evaluation of Proposed Method

Now that we have a database of previous (executed) experiments, the metric to measure distance between data sets: $dist_{DCT}$, and the metrics to measure distance between algorithms: $dist_{ONT\text{-}identity}$, $dist_{ONT\text{-}class}$, $dist_{ONT\text{-}property}$ and $dist_{ONT\text{-}pathLength}$; given a new (unexecuted) experiment e_z, we can:

1. Compute its distances to all the previous (executed) experiments
2. Find previous experiments that are nearest (having smallest distance) to it
3. Predict (classify) the unknown performance $P(e_z)$ as 'best' or 'rest' based on the majority performance of its k-nearest experiments neighbors.

We evaluate the effectiveness of our proposed method on our database of experiments with a leave-one-out validation. Due to the nature of our experimental setup, each experiment in our database has six other experiments that are using the same data set. Hence, when we do a prediction on a test experiment, we first exclude from its training set, all experiments that have the same data set as the test experiment. This is to ensure (to some degree) data independency between the training and the testing set in our evaluation.

In our evaluation, we investigate different weightings of α and β given to distDCT and distONT and measure their effectiveness at predicting algorithm performance. We also vary the distONT metric used: i.e. we use the metric $dist_{ONT\text{-}identity}$, $dist_{ONT\text{-}class}$, $dist_{ONT\text{-}property}$ and $dist_{ONT\text{-}pathLength}$ separately and note their effectiveness at predicting algorithm performance. For the metrics $dist_{ONT\text{-}class}$, $dist_{ONT\text{-}property}$ and $dist_{ONT\text{-}pathLength}$, all measures were taken on the inferred ontology.

We also vary the values of k used in our classification of algorithm performance and note the effectiveness of the different values of k in predicting algorithm performance. We compare the prediction accuracy of our proposed method with:

- the prediction accuracy of using information from data characteristics alone (i.e. α = 1)
- the prediction accuracy of using an algorithm's identity alone (i.e. α = 0 with $dist_{ONT\text{-}identity}$ as the algorithm distance metric). This is equal to using the 'prior' knowledge on algorithm performance distribution (presented in Table 1) to make the prediction – i.e. it systematically selects for a given algorithm the majority classification ('best' or 'rest') of the given algorithm in our data sets
- the prediction accuracy of using the majority rule (i.e. always predicting as 'rest').

Figure 1. The Classification Algorithm sub tree in the ontology

5 RESULTS AND DISCUSSIONS

In this section we present the results of our experiments at predicting algorithm performance using both data set characteristics and algorithm characteristics as features, evaluated on our database of executed experiments.

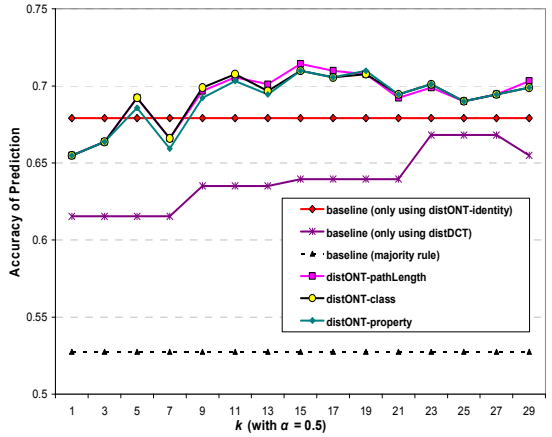


Figure 2. Prediction Accuracy at Different Values of k when $\alpha = 0.5$

Figure 2 shows the prediction accuracy at different values of k at $\alpha = 0.5$ (equal weights given to $dist_{DCT}$ and $dist_{ONT}$), using different $dist_{ONT}$ metrics compared to the three baselines.From Figure 2, it can be seen that when compared to the baseline of using information from data characteristics or algorithm identity alone, adding information from the ontology about algorithm characteristics improves the prediction accuracy at various values of k. The prediction accuracy is also significantly higher than the prediction accuracy of the majority rule baseline. The highest prediction accuracy at $\alpha = 0.5$ is achieved using $dist_{ONT\text{-}pathLength}$ metric at $k = 15$.

Using information on $dist_{ONT\text{-}identity}$ alone (which is equivalent to systematically select the majority classification ('best' or 'rest') of the given algorithm in our data sets) gives quite good prediction accuracy in our evaluation. This observation may be related to the fact that all the data sets used in our testing and training come from a specific domain (cancer, biology). Since all our data sets come from the same domain, they may possess certain *implicit* similar characteristics which make some algorithms consistently perform well on them. Hence, using 'prior' knowledge on algorithm performance distribution in this data domain (Table 1) may, in this case, be good enough to predict algorithm performance. However, when a test experiment involves a data set that comes from a different domain than our current domain, it may not always be good enough to simply rely on the algorithm identity metric to predict algorithm performance on this new data set since the data set may come from a domain which has a totally different algorithm performance distribution than our current domain.

Figure 2 also highlights an important issue with using only data set characteristics (without knowledge of the algorithm) to predict the performance of algorithms. From Figure 2, it is clear that data set characteristics cannot stand alone; they must be tied to the algorithm in predicting performance. Predicting performance based on data set characteristics alone gives lower accuracy. Furthermore, if data set characteristics measured are not relevant to the success or failure of the algorithm being investigated on the data set, this may result in a misleading or even worse prediction. Since there are inevitably many ways to describe a data set, coming up with a reliable data set descriptors (descriptors that are relevant to the success or failure of a particular algorithm on the data set) may prove to be a daunting task. The advantage of our proposed method is that we make up for what we possibly lack in our knowledge of data set characteristics with our knowledge of algorithm characteristics (and vice versa) in order to make better prediction of algorithm performance.

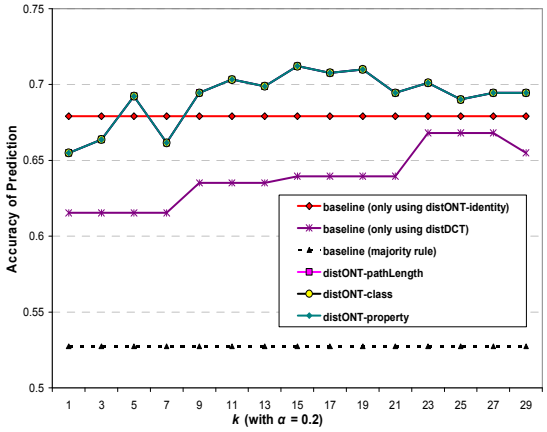


Figure 3. Prediction Accuracy at Different Values of k when $\alpha = 0.2$

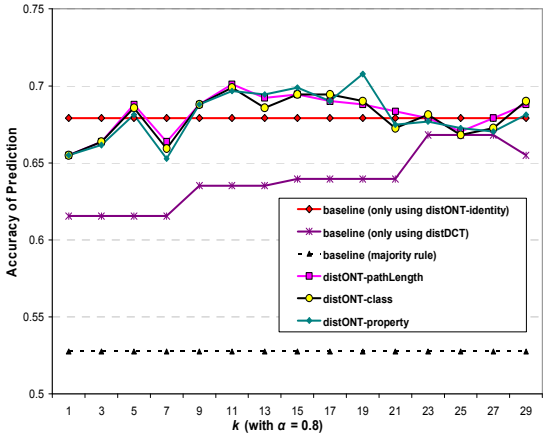


Figure 4. Prediction Accuracy at Different Values of k when $\alpha = 0.8$

Figure 3 and 4 show the prediction accuracy at different values of k when $\alpha = 0.2$ (more weight given to $dist_{ONT}$), and $\alpha = 0.8$ (more weight given to $dist_{DCT}$) respectively. Figure 3 and 4 show similar trends to Figure 2. Similar to Figure 2, it can be seen that at different values of k, adding information about algorithm characteristics from the ontology improves the prediction accuracy when compared to using data set characteristics or algorithm identity alone or using majority rule.

An interesting trend to notice is that at lower value of α (more weight given to $dist_{ONT}$) the various $dist_{ONT}$ metrics performances converge. This could indicate that the various $dist_{ONT}$ metrics that we use actually measure the same aspects of algorithm similarity and whatever subtle differences between them are insignificant when they are given more weight than $dist_{DCT}$ in predicting performance. This further indicates that there may be a complex interplay between our choice of $dist_{ONT}$ metrics and our choice of $dist_{DCT}$. Further investigation is necessary to understand this phenomenon.

6 CONCLUSIONS & FUTURE WORK

In this paper we present a preliminary study on using both information on data set characteristics and algorithm characteristics as features to predict algorithm performance on the data set. Our assumption is that *algorithms with similar characteristics will have similar performance on data sets with similar characteristics*. We compute data set characteristics using geometrical descriptors of class boundary [2]. We compute algorithm characteristics and similarities using kernel similarity methods [6] on our data mining ontology [5].

We evaluate our proposed method through a set of experiments conducted on cancer data sets. We report an increase in prediction accuracy when using both data set characteristics and algorithm characteristics as features, compared to using data set characteristics or algorithm characteristics alone. This is despite the fact that we only use simple metrics (identity, common class, property and path length) to compute algorithm characteristics on the ontology.

Our study also highlights issues in using data set characteristics alone or algorithm characteristics alone to predict algorithm performance. To predict algorithm performance, data set characteristics cannot be used alone (without regards to the algorithm). They must be tied to the algorithm investigated to predict its performance. Algorithm performance distribution, on the other hand, is highly domain-dependent. To make a good prediction of algorithm performance across different domains, the characteristics of the algorithm investigated must be tied to the characteristics of the data set on which its performance is to be predicted. Our proposed method combines both knowledge of data set characteristics and algorithm characteristics; thus improving the prediction accuracy.

The improvement in prediction accuracy suggests the merit of our proposed method and the potential for improving it even further using metrics that can exploit even more the information on algorithm characteristics in the ontology. For example, we can compute finer-grained similarity of algorithm object properties by following further dependencies from concepts referenced by object properties and compute recursively their kernel similarities.

In the future, we can also predict algorithm performance in two steps (instead of in one step as we do here). We can first query the nearest neighbours of an experiment based on the data distance ($dist_{DCT}$) alone and then query the resulting nearest neighbours based on the algorithm distance ($dist_{ONT}$) alone (and vice versa). Using two steps, we can control and observe better the effect of each distance measures in our prediction instead of just using simple flat weighting scheme (α and β) that we use in this study.

In the future, it will also be interesting to formulate our problem of predicting algorithm performance as a multi-class labeling problem in which performance is measured not as an absolute, but as a relative performance between algorithms [12].

Further, in this study we only predict algorithm performance on a data set for the task of classification. In the future, our work can be extended to predicting algorithm performance on a data set for tasks other than classification. We can also extend it even further to predict the performance of a sequence of algorithms on a data set – e.g. in predicting the performance of data mining workflows.

ACKNOWLEDGEMENTS

This work was partially supported by the European Union within FP7 ICT project e-LICO (Grant No 231519).

REFERENCES

[1] D. Michie, D.J. Spiegelhalter, and C. C. Taylor. *Machine Learning, Neural and Statistical Classification*, Ellis Horwood Upper Saddle River, NJ (1994).

[2] T.K. Ho and M. Basu. Complexity Measures of Supervised Classification Problems. *IEE Transactions on Pattern Analysis and Machine Intelligence,* 24(3):289-300 (2002).

[3] T.K. Ho and M. Basu. *Data Complexity in Pattern Recognition (Advanced Information and Knowledge Processing)*, Springer-Verlag New York, Inc., Secaucus, NJ (2006).

[4] e-LICO Project. 29 January 2009. 13 May 2010. <http://www.e-lico.eu/>.

[5] M. Hilario, A. Kalousis, P. Nguyen, and A. Woznica. A Data Mining Ontology for Algorithm Selection and Meta Learning. In: *Proc of the ECML/PKDD09 Workshop on Third Generation Data Mining: Towards Service-Oriented Knowledge Discovery* (SoKD-09), Bled, Slovenia (2009).

[6] S. Bloehdorn and Y. Sure. Kernel Methods for Mining Instance Data in Ontologies. In: *Proc of the 6th International and 2nd Asian Semantic Web Conference* (ISWC 2007 and ASWC 2007), Busan, South Korea (2007).

[7] B. Pfahringer, H. Bensusan, C. Giraud-Carrier. Meta-learning by Landmarking Various Learning Algorithms. In: *Proc of the 17th International Conference on Machine Learning* (ICML 2000), Stanford, CA, USA (2000).

[8] T.M. Cover and P.E. Hart. Nearest Neighbor Pattern Classfication. *IEEE Transactions on Information Theory, 31(1):21-27* (1967).

[9] Resources: Cancer Data Sets. 17 May 2010. 17 May 2010. <http://derrywi.byethost15.com/index.htm#progress>.

[10] Rapid-I. 29 May 2007. 13 May 2010. <http://rapid-i.com/>.

[11] G.W. Corder and D. I. Foreman, *Nonparametric Statistics for Non-Statisticians: A Step-by-Step Approach*, Wiley, NJ (2009).

[12] L. Todorovski, H. Blockeel, and S. Dzeroski. Ranking with Predictive Clustering Trees. In: *Proc of the 13th European Conference on Machine Learning* (ECML 2002), Helsinki, Finland (2002).

On the importance of similarity measures for planning to learn

Hendrik Blockeel [1,2] and **Hossein Rahmani** [2] and **Tijn Witsenburg** [2]

Abstract. Data analysis is a complex process that consists of finding a suitable data representation, a suitable machine learning method, and using a suitable evaluation metric (one that reflects what the user is really interested in). All these choices are crucial from the "planning to learn" perspective, and none are trivial. In this paper we focus on the first of these three, the input space representation. Sometimes this problem is posed as "defining the right features", but in those cases where we have non-standard data, for instance, for relational or graph data, the data representation problem does not map easily on feature construction. In some sense, it is easier to see it as a problem of constructing a suitable distance metric, similarity metric, or kernel. In this paper we discuss one particular way of defining a similarity measure in the context of annotated graphs. We illustrate the importance of the choice of distance measure with an experiment on the Cora dataset.

1 INTRODUCTION

It is generally agreed that the Knowledge Discovery process comprises much more than the data mining step alone [1]. The user first needs to understand the knowledge discovery problem, the goals of the knowledge discovery task, and the data that will be used as input. Based on this insight, more technical choices need to be made regarding:

- The data representation, or input space representation. This concerns how the input data for the learner are represented. We prefer the term "input space representation" here because "data representation" may suggest that this question relates to how single data elements are represented. In fact, as we will argue further on, the question is not so much how to represent individual data elements, but how to represent the data set as a whole, or how to represent the relative positioning of the elements with respect to each other and with respect to the entire input space.
- The machine learning methods to be used. Obviously, different tasks call for different machine learning methods. But even among methods that fulfill the same task (for instance, classification), each method has its particular bias [3]. This bias, however, is only defined relative to the concrete input space being used. In a concrete application context, the bias of the machine learning process is determined by both the input representation and the learning method being used.
- The evaluation metrics: these are directly related to the goals of the knowledge discovery process. They are in a sense a formal translation of the user's success criteria to the context of the pattern language used to describe the output of the learning system.

[1] Department of Computer Science, Katholieke Universiteit Leuven
[2] Leiden Institute of Advanced Computer Science, Leiden University

In this paper, we investigate in some more detail the data representation question. We focus on the particular case of learning from annotated graphs; these are graphs where nodes, edges, or whole (sub)graphs can have additional information attached to them. We focus on the case of graphs with annotated nodes, and discuss different ways in which similarity measures can be defined for those graphs. We present experimental results showing that the choice of the distance measure indeed has a significant influence on the results of the classification process, which suggests that this issue may deserve more attention.

2 ANNOTATED GRAPHS

With an "annotated graph" we mean a graph structure where nodes, edges, (sub)graphs can be annotated with additional information. That is, an annotated graph is a structure (V, E, λ) with V the node set, $E \in V \times V$ a set of edges, and $\lambda : 2^V \rightarrow \mathcal{A}$ a function that assigns to any subset S of V an "annotation" $a \in \mathcal{A}$. The space of possible annotations is left open; it can be a set of symbols from, or strings over, a finite alphabet; the set of reals; an n-dimensional Euclidean space; a powerset of one of the sets just mentioned; etc. When S is a single node or a pair of two nodes (i.e., an edge), we say that the annotation is attached to the node or edge.

In the remainder of this text we will consider the more specific case of graphs where only nodes are annotated. The annotations will typically be treated as vectors or subsets from a predefined set (note that such a subset can be represented as a boolean vector with one component per possible element, which has the value true if and only if the element is in the subset).

Note that if the edges in this graph structure are ignored, the dataset is reduced to a set of nodes V with each node annotated with a vector and each node standing on its own; since the nodes carry no information except for the vector attached to them, we essentially obtain a standard attribute-value dataset.

Regarding the learning task, we focus on the case where single nodes are the data elements that have to be classified or clustered. Classification or clustering can be seen as completing the annotation of a node by filling in the value for one particular component of the annotation, the so-called target variable, which indicates the class or cluster the element belongs to.

3 THE ROLE OF SIMILARITY MEASURES IN LEARNING

Clustering processes typically depend on a notion of similarity (sometimes expressed using distance metric) between the elements to be clustered. Also classification processes rely on such a notion,

sometimes explicitly, sometimes implicitly. In instance-based learning, such as k-nearest-neighbor methods, the reliance on a similarity measure is explicit; more specifically, these methods rely on the assumption that instances that are similar tend to belong to the same class. (In the regression context, the corresponding assumption is that the function to be learned is continuous, which is often a reasonable assumption.) Also kernel-based methods rely on the notion of similarity, since a kernel function can be interpreted as indicating the similarity between two points. But even a decision tree learner, for instance, which is normally not considered a similarity-based method, can be interpreted in those terms. A decision tree essentially expresses the hypothesis that whenever two instances have the same values for a limited number of attributes (so that they end up in the same leaf), they are similar enough to belong to the same class. Learning a decision tree corresponds to identifying what the relevant combinations of attributes are, on the basis of which similarity can be decided; in other words, it corresponds to learning a suitable similarity measure. A similar claim can be made for rule learners.

In this sense, it is clear that the bias of a learning method is strongly connected to the notion of a similarity measure in the input space. From the point of view of planning to learn, we believe this is an important observation. It is generally accepted that bringing the input data into the right format (defining the right features, for instance), is an important step on which the performance of a learner will crucially depend. But the notion of similarity is equally important: whereas an incorrect representation may make it impossible for a learner to use a correct similarity measure (e.g., for a decision tree, not having the right features makes it impossible for the tree to express the correct combinations of features), having a correct representation does not guarantee that the learner will be able to use the right similarity measure.

Rather than stating that an input format is suitable for learning if it contains the right features, which corresponds to stating that the input space has the right dimensions, it may be more accurate to state that the input space is right if individual elements are embedded into a metric space in such a way that elements that are more likely to have the same class are close to each other, or similar, according to some definition of similarity.

This view is quite similar to the view taken in kernel-based methods, where the kernel implicitly defines such an embedding into a metric space. The point that we are making here, is that, from the point of meta-learning or planning to learn, it may be useful to may more attention to construction the right similarity measure, rather than simply constructing the right features.

To make this a bit more concrete: suppose we want to learn the function A XOR B. It is not possible to state whether a set of features is the right one, without considering it together with the learner that will be used; conversely, it is not possible to state whether a learner will be suitable without considering the input features. For a decision tree learner, an input space with features $\{$A,B$\}$ allows the learner to construct a correct decision tree, but for a single-layer perceptron the same input space is not sufficient. If we focus on the notion of similarity, we can say that any embedding of the data into an input space that considers two points to be similar if their A XOR B value is the same, is a good embedding. The notion of similarity can thus replace the notion of "combination of input space and learner", and offers in a sense a more principled description of the latter. Stated in yet another way, there is no need to consider different learners; a single instance-based learner, making use of the right similarity measure, can mimic the learning behavior of any other learning method.

4 SIMILARITY MEASURES FOR NODES IN GRAPHS

The above discussion is particularly relevant in the context of learning from annotated graphs. The information contained in these graphs is not easily embedded in a finite-dimensional space (i.e., it is not possible to define, in general, a finite set of features that represents all the information in the graph, and which can be used for learning). However, as argued before, what is really relevant is simply the notion of similarity. More specifically, when clustering or classifying nodes, if we can define the right similarity measure for these nodes, we will obtain a good clustering, or accurate classification.

This brings us to the question of defining similarity measures for nodes in an annotated graph. The following discussion is based on earlier work by Witsenburg et al. [5].

Concerning clustering in the context of graphs, we can distinguish "standard clustering" algorithms and "graph clustering" algorithms [5]. In **standard clustering**, items are clustered according to their similarity whilst not taking into account any relational information (that is, all edges in the graph are simply ignored). A distance or similarity function is given that for any pair of items tells us how similar they are. In this way a $N \times N$ matrix can be created with all these values. In **graph clustering**, unlabeled graphs are considered, where clustering (or partitioning) the graph typically means finding subgraphs of the graph such that the number of links connecting different subgraphs is as small as possible whilst not taking into account any information about the content of the node.

Any $N \times N$ matrix can be converted to a graph (with the element A_{ij} representing the weight of the edge between nodes i and j) and vice versa. This raises the question whether, in those cases where both node content and graph structure are available (such as the Web), one could find a clustering method that combines both types of information.

Neville et al. (2003) discuss this problem, and discuss a number of possible solutions. In the combined method they propose, the structure of the graph remains the same; the edges of the graph are given weights that correspond to the similarity between the nodes they connect, then a graph clustering algorithm is applied to them. Neville et al. compare different graph clustering algorithms.

Witsenburg et al. propose an opposite direction: instead of introducing the content information in the graph (in the form of edge weights that indicate the similarity between nodes), they inject the structure information into the similarity function, after which a standard clustering algorithm is used. One could say that Neville et al. map the hybrid clustering task onto graph clustering, whereas Witsenburg et al. map it onto standard clustering.

In the following section we discuss three similarity measures that fit into Witsenburg's framework.

5 SIMILARITY MEASURES: CONTENT-BASED, STRUCTURE-BASED, HYBRID

Consider a graph $G(V, E, \lambda)$. We distiguish the following types of similarity measures.

Content-based measures: a content-based similarity measure $S : \mathcal{A} \times \mathcal{A} \rightarrow \mathbb{R}$ is one that compares two nodes by looking at their annotations (or "contents"), not taking their environment (the graph structure) into account.

Structure-based, or contextual, measures: a structure-based similarity measure is one that takes the context of the nodes into ac-

count, but not the contents of the nodes themselves (though it may be exploiting the contents of neighboring nodes, for instance). In the framework described above, this implies that a structure-based similarity measure $S' : \mathcal{A}' \times \mathcal{A}' \rightarrow \mathbb{R}$ implies that the context of a node can be mapped into a space $\mathcal{A}'$, which can be seen as an alternative annotation; this annotation contains all the structural information about the node that will be used to determine similarities; once the mapping has been made, each node is treated as an independent point in the input space. (In the terminology of propositional and relational learning, this process is called "propositionalisation").

Hybrid measures: these are measures that combine both content and structure information.

In all three of the above cases, a similarity matrix M can be constructed, where M_{ij} indicates the similarity between node i and node j. From here on, we will denote with M the similarity matrix corresponding to the content-based similarity measure S, and with M' the similarity matrix corresponding to the structure-based measure S'. In this context we are assuming that there is a natural similarity measure defined over $\mathcal{A}$; what we are interested in, is ways to define M'.

M' will be using structural information, and this structural information is naturally encoded in the adjacency matrix A of the graph G. One way to define M' is the following equation:

$$M' = M \times A + A \times M \tag{1}$$

or, in terms of individual elements,

$$m'_{ij} = \sum_{n=0}^{N} (m_{in} \cdot a_{nj}) + \sum_{n=0}^{N} (a_{in} \cdot m_{nj}) \tag{2}$$

The practical meaning of the values in M' can easily be understood when taking a closer look at equation 2. Considering the first part ($\sum_{n=0}^{N}(m_{in} \cdot a_{nj})$) for every a_{nj} it holds that it is 0 when there is no relation between node n and node j and 1 otherwise. Thus, this first part will sum all values m_{in} for which a_{nj} is equal to 1. This can be described by saying that the first part gives the sum of all similarities of node i to all neighbours of j in the graph describing the relations. Analogously the second part is the sum of all similarities of node j to all neighbours of i.

In some cases, the above formula is counterintuitive. By summing similarities of neighbours, we get the effect that nodes with high degree will tend to score higher with respect to their similarity in S' than nodes with low degree. To counter this effect, we can use average values instead of sum; to that effect, equation 2 needs to be changed into equation 3.

$$m'_{ij} = \frac{1}{2} \cdot \left(\frac{\sum_{n=0}^{N}(m_{in} \cdot a_{nj})}{\sum_{n=0}^{N} a_{nj}} + \frac{\sum_{n=0}^{N}(a_{in} \cdot m_{nj})}{\sum_{n=0}^{N} a_{in}} \right) \tag{3}$$

The constant '$1/2$' in equation 3 ensures that all values in M' are in the same range as the values in M.

In the remainder, we will denote with M' the matrix defined according to equation 3.

We now have a similarity measure S that looks only at the content (annotations) of nodes to compare them, and a measure S' that looks at the content of neighboring nodes to compare two nodes, but never compares the content of the nodes directly. One could also combine the two into a hybrid form; a natural way of combining them is taking the average of the two:

$$M" = (M + M')/2 \tag{4}$$

The definitions of M' and $M"$ are rather ad hoc; many other ways of defining similarity measures for nodes in a graph can be defined. We are not trying to argue here that these definitions are useful, or under which conditions they are useful. The point that we are trying to make is that multiple definitions of similarity may make sense, and that depending on the task, one may be more suitable than the other. In fact, this effect may be visible even within a single dataset, as our experiments will show. The same experiments will also show that the similarity measure may influence the results of a learning procedure in some systematic way.

6 EXPERIMENTS

6.1 Cora Data Set

We have experimented with the well-known Cora data set [2]. This is a graph-structured data set containing 37,000 nodes; each node represents a scientific paper; two nodes are linked if they cite each other (the links are undirected: it is not known which is the citing and cited paper); each node is annotated with the abstract of the paper it represents (more precisely: with a bag-of-words representation of this abstract), and with a label denoting one or more subject classes (from 70 possible classes).

The content-based similarity S is defined over the bag-of-words only, not over the class labels. The matrix elements of M are computed as follows:

$$m_{ij} = \frac{1}{2} \cdot \left(\frac{|b_i \cap b_j|}{|b_i|} + \frac{|b_i \cap b_j|}{|b_j|} \right) \tag{5}$$

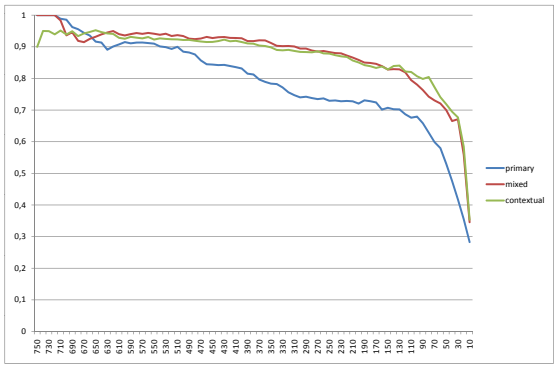
with b_i the bag of words corresponding to node i. It can easily be seen that this is the average ratio of words that are in common between two papers. The adjacency matrix A was created by taking into account the citation relation: a_{ij} is 1 when paper i cites paper j or is cited by it, and 0 otherwise.

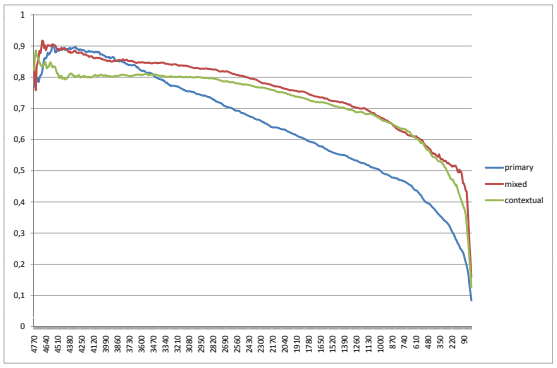
Five subsets of the Cora dataset were created. Each subset is clustered with agglomerative hierarchical clustering: once with the content based (primary) similarity, once with the contextual (secundary) similarity and once with the hybrid (mixed) similarity. On the x-axis is the amount of clusters, which goes from $|V|$ to 1 following the working method of agglomerative hierarchical clustering.

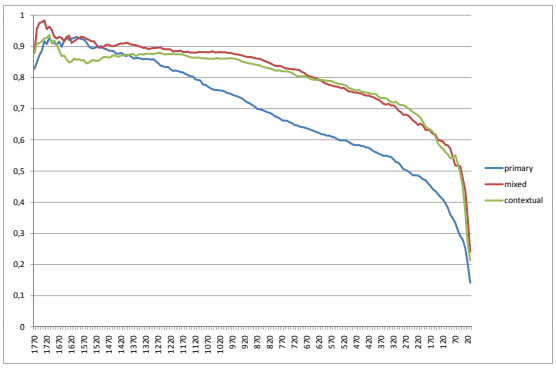
Every ten iterations the quality of the clustering is calculated. To do so, only clusters with size greater than one are considered. For these clusters the quality is calculated by taking the average Jaccard index of the classes of each pair of elements in this cluster. More specifically, if we have two nodes i and j, each of which are labelled with a set of classes c_i and c_j, their Jaccard index is $|c_i \cap c_j|/|c_i \cup c_j|$. The quality of a clustering is the average Jaccard index between any two nodes in the same cluster, averaged over all clusters.

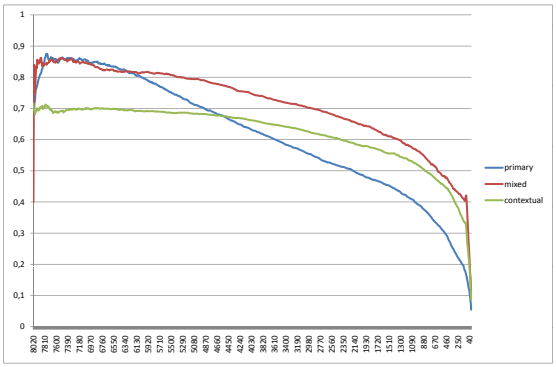
The graphs clearly show some remarkable effects for which we do not have a straightforward explanation, but the existence of which we find interesting. Starting with many small clusters, the contextual similarity measure performs less well than the content-based similarity measure (that is, the clusters it forms are less coherent with respect to the classes of their elements). As the number of clusters goes down and the clusters become larger, however, there is a point where the contextual similarity starts performing better than the content based similarity.

Another remarkable fact is that the hybrid similarity performs well globally: it appears quite successful at combining the advantages of both other similarity measures.









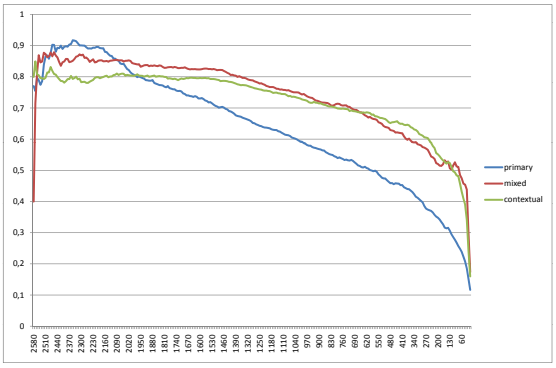


Figure 1. Evolution of clustering quality in terms of number of clusters. While content-based similarity yields better (more coherent) clusters when many small clusters are produced, contextual similarity performs better when fewer large clusters are produced. The hybrid similarity measure tends to globally outperform or coincide with the best of these two.

7 CONCLUSIONS

In this paper we have argued that, while it is generally accepted that the input representation influences the quality of the learning process, it may be better to study input representation in terms of the similarity measure that is defined over the input space, rather than in terms of the features that are being used. In the context of learning from graphs (but also in other contexts), it may be possible to define such a similarity measure without specifying precisely the features on which this similarity measure is based. With some experiments on the Cora dataset, we have shown that the similarity measure that one uses indeed influences the results, and moreover, in this case it appears to influence them in a systematic way. This suggests that the influence of similarity measures on learning performance can, and should, be studied. Up till now, little attention has been paid to this particular aspect of learning bias. It may be useful to increase that attention in the future.

ACKNOWLEDGEMENTS

This research is funded by the Dutch Science Foundation (NWO) through a VIDI grant.

REFERENCES

[1] P. Chapman, J. Clinton, R. Kerber, T. Khabaza, T. Reinartz, C. Shearer, and R. Wirth, 'CRISP-DM 1.0: Step-by-step data mining guide', Technical report, CRISP-DM Consortium, (2000).

[2] A. McCallum, K. Nigam, J. Rennie, and K. Seymore, 'Automating the construction of internet portals with machine learning', *Information Retrieval*, **3**, 127–163, (2000).

[3] T. Mitchell, *Machine Learning*, McGraw-Hill, 1997.

[4] J. Neville, M. Adler, and D. Jensen, 'Clustering relational data using attribute and link information', in *Proceedings of the Text Mining and Link Analysis Workshop, 18th International Joint Conference on Artificial Intelligence*, (2003).

[5] T. Witsenburg and H. Blockeel, 'A method to extend existing document clustering procedures in order to include relational information', in *Proceedings of the 6th International Workshop on Mining and Learning with Graphs*, pp. 1–3, (2008).

A Similarity-based Adaptation of Naive Bayes for Label Ranking: Application to Metalearning for Algorithm Selection

Artur Aiguzhinov [1] and **Carlos Soares** [2] and **Ana Paula Serra** [3]

Abstract.
The problem of learning label rankings is receiving increasing attention from several research communities. A number of common learning algorithms have been adapted for this task, including k-Nearest Neighbors (kNN) and decision trees. Following this line, we propose an adaptation of the naive Bayes classification algorithm for label ranking problem. Our main idea lies in the concept of similarity between the rankings measured by a distance function. We empirically test the proposed method on some metalearning problems that consist of relating characteristics of learning problems to the relative performance of learning algorithms. Our method generally performs better than the baseline indicating that it is able to identify some of the underlying patterns in the data.

1 Introduction

Label ranking is an increasingly popular topic in the machine learning literature [5, 2]. Label ranking studies a problem of learning a mapping from instances to rankings over a finite number of predefined labels. It can be considered as a natural generalization of the conventional classification problem, where only a single label is requested instead of a ranking of all labels [2]. In contrast to a classification setting, where the objective is to assign examples to a specific class, in label ranking we are interested in assigning a complete preference order of the labels to every example.

Several methods have been developed for label ranking some of which consist of adapting existing classification algorithms (e.g., k-Nearest Neighbor [1], decision trees [11]. In this paper, we propose an adaptation of the naive Bayes (NB) algorithm for label ranking. One possible approach to adapt NB for ranking is based on probabilistic models for ranking (e.g., [6]). However, we follow a different approach. Given that the accuracy of label ranking methods is often measured using the correlation between the predicted and target rankings (e.g., using Spearman's rank correlation [1]), we believe that it is best to align the algorithm with the evaluation measure. Therefore we propose the adaptation of the NB algorithm for label ranking based on correlation.

[1] Faculty of Economics, University of Porto, Portugal, email: artur@liaad.up.pt
[2] LIAAD-INESC LA, Rua de Ceuta 118-6, 4050-190, Porto, Portugal, email: csoares@fep.up.pt
[3] CEFUP - Centro de Economia e Finanças da Universidade do Porto, email: aserra@fep.up.pt

The paper is organized as follows: section 2 provides the formalization of the label ranking problem; section 3 explains the problem of metalearning, which will be the application domain for the empirical evaluation; section 4 briefly describes the naive Bayes algorithm for classification; section 5 shows the adaptation of the NB algorithm for label rankings; section 6 presents the results of applying the adapted NB algorithm,; and section 7 concludes with the goals for future work.

2 Learning label rankings

Based on [12], a label ranking problem is defined as follows. Let $\mathcal{X} \subseteq \{\mathcal{V}_1, \ldots, \mathcal{V}_m\}$ be an instance space of nominal variables, such that $\mathcal{V}_a = \{v_{a1}, \ldots, v_{an_a}\}$ is the domain of nominal variable a. Also, let $\mathcal{L} = \{\lambda_1, \ldots, \lambda_k\}$ be a set of labels, and $\mathcal{Y} = \Pi_{\mathcal{L}}$ be the output space of all possible total orders[1] over $\mathcal{L}$ defined on the permutation space Π. The goal of a label ranking algorithm is, given a training set $\mathcal{T} = \{x_i, y_i\}_{i \in \{1,\ldots,n\}} \subseteq \mathcal{X} \times \mathcal{Y}$ of n examples, to learn a mapping $h : \mathcal{X} \to \mathcal{Y}$, where h is chosen from a given hypothesis space $\mathcal{H}$, such that a predefined loss function $\ell : \mathcal{H} \times \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ is minimized.

Unlike classification, where for each instance $x \in \mathcal{X}$ there is an associated class $y_i \in \mathcal{L}$ in label ranking problems, there is a ranking of the labels associated with every instance x and the goal is to predict it. This is also different from other ranking problems, such as in information retrieval or recommender systems. In these problems the target variable is a set of ratings or binary relevance labels for each item, and not a ranking.

The algorithms for label ranking can be divided into two main approaches: methods that transform the ranking problem into multiple binary problems and methods that were developed or adapted to predict the rankings. An example of the former is the ranking by pairwise comparisons [5]. Some examples of algorithms that are specific for rankings are: the predictive clustering trees method [11], the similarity-based k-Nearest Neighbor for label ranking [1], the probabilistic k-Nearest Neighbor for label ranking [2] and the linear utility transformation method [4, 3].

[1] A total order is a complete, transitive, and asymmetric relation $\succ_x$ on $\mathcal{L}$, where $\lambda_i \succ_x \lambda_j$ indicates that λ_i precedes λ_j. In this paper, given $\mathcal{L} = \{A, B, C\}$, we will use the notation $\{A, C, B\}$ and $\{1, 3, 2\}$ interchangeably to represent the order $A \succ_x C \succ_x B$.

To assess the accuracy of the predicted rankings relative to the corresponding target rankings a suitable loss function is needed. In this paper we use the Spearman correlation coefficient [1, 12]:

$$\rho(\pi_i, \hat{\pi}_i) = 1 - \frac{6 \sum_{k=1}^{d} (\pi_i - \hat{\pi}_i)^2}{k^3 - k} \quad (1)$$

where π and $\hat{\pi}$ are, respectively, the target and predicted rankings for a given instance. Two orders with all the labels placed in the same position will have a Spearman correlation of +1. Labels placed in reverse order will produce correlation of -1. Thus, the higher the value of ρ the more accurate the prediction is compared to target. An extensive survey of label ranking algorithms is given by [12].

3 Metalearning

Many different learning algorithms are available to data analysts nowadays. For instance, decision trees, neural networks, linear discriminants, support vector machines among others can be used in classification problems. The goal of data analysts is to use the one that will obtain the best performance on the problem at hand. Given that the performance of learning algorithms varies for different datasets, data analysts must select carefully which algorithm to use for each problem, in order to obtain satisfactory results.

Therefore, we can say that a performance measure establishes a preference relation between learning algorithms for each problem. For instance, Table 1 illustrates the preference relations between four classification algorithms (a_i) on two datasets (d_j) defined by estimates of the classification accuracy of those algorithms on those datasets.

	a_1	a_2	a_3	a_4
d_1	90%	61%	82%	55%
d_2	84%	86%	60%	79%

Table 1. Accuracy of four learning algorithms on two classification problems.

Selecting the algorithm by trying out all alternatives is generally not a viable option. As explained in [11]:

> In many cases, running an algorithm on a given task can be time consuming, especially when complex tasks are involved. It is therefore desirable to be able to predict the performance of a given algorithm on a given task from description and without actually running the algorithm.

The learning approach to the problem of algorithm recommendation consists of using a learning algorithm to model the relation between the characteristics of learning problems (e.g., application domain, number of examples, proportion of symbolic attributes) and the relative performance of a set of algorithms [1]. We refer to this approach as *metalearning* because we are learning about the performance of learning algorithms.

Metalearning approaches commonly cast the algorithm recommendation problem as a classification task. Therefore, the recommendation provided to the user consists of a single algorithm. In this approach, the examples are datasets and the classes are algorithms. However, this is not the most suitable form of recommendation. Although the computational cost of executing all the algorithms is very high, it is often the case that it is possible to run a few of the available algorithms. Therefore, it makes more sense to provide recommendation in the form of a ranking. The user can then execute the algorithms in the suggested order, until no computational resources (or time) are available. The data provided to the algorithm for learning rankings consists of two matrices. The first one contains a set of variables that describe the dataset. In the case of metalearning, it often contains variables that represent general and statistical properties of the datasets. The second matrix contains the target rankings, based on the performance of the algorithms on the datasets.

4 The Naive Bayes Classifier

We follow [7] to formalize the naive Bayes classifier. In classification each instance $x \in \mathcal{X}$ is binded to class $y_i \in \mathcal{L}$. The task of a learner is to create a classifier C from the training set $\mathcal{T}$. The classifier takes a new, unlabeled instance and assigns it to a class (label).

The Naive Bayes method classifies a new instance x_i by determining the most probable target value, $c_{MAP}(x_i)$[2], given the attribute values that describe the instance.

$$c_{MAP(x_i)} = \arg\max_{\lambda \in \mathcal{L}} P(\lambda | x_{i,1}, x_{i,2}, \ldots, x_{i,m}) \quad (2)$$

where $x_{i,j}$ is the value of attribute j for instance i.

The Bayes theorem establishes the probability of A given B as:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \quad (3)$$

Thus, the Bayes theorem provides a way to calculate the probability of a hypothesis based on: i) its prior probabilities, ii) observed data, and iii) probabilities of observing various data given the hypothesis.

We can rewrite (2) as

$$\begin{aligned} c_{MAP(x_i)} &= \arg\max_{\lambda \in \mathcal{L}} \frac{P(x_{i,1}, x_{i,2}, \ldots, x_{i,m} | \lambda) P(\lambda)}{P(x_{i,1}, x_{i,2}, \ldots, x_{i,m})} \\ &= \arg\max_{\lambda \in \mathcal{L}} P(x_{i,1}, x_{i,2} \ldots x_{i,m} | \lambda) P(\lambda) \end{aligned} \quad (4)$$

The naive Bayes classifier makes one simple, hence, naive assumption that the attribute values are conditionally independent from each other. This implies that the probability of observing the conjunction $x_{i,1}, x_{i,2}, \ldots, x_{i,m}$ is the product of the probabilities for the individual attributes: $P(x_{i,1}, x_{i,2}, \ldots, x_{i,m} | \lambda) = \prod_{j=1}^{m} P(x_{i,j} | \lambda)$. Substituting this expression into equation (4), we obtain the Naive Bayes Classifier:

$$c_{nb}(x_i) = \arg\max_{\lambda \in \mathcal{L}} P(\lambda) \prod_{j=1}^{m} P(x_{i,j} | \lambda) \quad (5)$$

[2] MAP – Maximum A Posteriori

5 Adapting NB to Ranking

Consider the classic problem of the play/no play tennis based on the weather conditions. The naive Bayes classification algorithm can be successfully applied to this problem [7, chap. 6]. For illustration purposes, we extend this example application to the label ranking setting by replacing the target with a ranking on the preferences of a golf player regarding 3 tennis courts on different days (Table 2). The last three columns in Table 2 represent the ranks of the tennis courts A, B and C.

Table 2. Example of tennis courts $\{A, B, C\}$ rankings based on the observed weather conditions

Day	Outlook	Temp	Hum	Wind	Ranks		
					A	B	C
1	Sunny	Hot	High	Weak	1	2	3
2	Sunny	Hot	High	Strong	2	3	1
3	Overcast	Hot	High	Weak	1	2	3
4	Rain	Mild	High	Weak	1	3	2
5	Rain	Mild	High	Strong	1	2	3
6	Sunny	Mild	High	Strong	2	3	1

The adaptation of the NB algorithm for label ranking requires the adaptation of the following concepts:

- prior probability, $P(y)$
- conditional probability, $P(x|y)$

The adaptation is required due to the differences in nature between label rankings and classes. The prior probability of ranking $\{A, B, C\}$ on the data given in Table 2 is $P(\{A, B, C\} = 3/6 = 0.5$, which is quite high. On the other hand, the probability of $\{A, C, B\}$ is quite low, $P(\{A, C, B\} = 1/6 = 0.167$. However, taking into account the stochastic nature of these rankings [2], it is intuitively clear that the observation of$\{A, B, C\}$ increases the probability of observing $\{A, C, B\}$ and vice-versa. This affects even rankings that are not observed in the available data. For example, the case of unobserved ranking $\{C, B, A\}$ in Table 2 would not be entirely unexpected in the future considering that similar observed ranking $P(\{C, A, B\} = 2/6 = 0.33$.

One approach to deal with this characteristic of label rankings is to use ranking distributions, such as the Mallows model (e.g., [6, 2]). Alternatively, we may consider that the intuition described above is motivated by the varying similarity between rankings. One method that follows this approach is the k-NN ranking algorithm [1].

Distance-based label ranking algorithms have two important properties:

- they assign non-zero probabilities even for rankings which have not been observed. This property is common to distribution-based methods.
- the algorithms are based on the notion of similarity between rankings, which also underlies the evaluation measures that are commonly used. Better performance is naturally expected by aligning the algorithm with the evaluation measure.

In order to adapt NB for label ranking based on the concept of similarity, it is necessary to establish a parallel between the similarity of rankings and the concept of likelihood, which underlies this algorithm. This parallel has been established for the general Euclidean distance measure [13]. Although not all assumptions required for that parallel hold when considering the distance between rankings, given that the naive Bayes algorithm is known to be robust to violations of its assumptions, we propose a similarity-based adaptation of NB for label ranking.

Define $\mathcal{S}$ as a similarity matrix between the target rankings in a training set, i.e. $\mathcal{S}_{n\times n} = \rho(y_i, y_j)$. The *prior likelihood* of a label ranking (concept equivalent to the prior probability in classification) is given by:

$$L(y) = \frac{\sum_{i=1}^{n} \rho(y, y_i)}{n} \tag{6}$$

We say that the prior likelihood is the measure of similarity of each of the rankings to the total list of rankings. We measure similarity using the Spearman correlation coefficient. Equation 6 shows the average distance of one rank relative to others. The greater the similarity between two particular rankings, the smaller is the distance between them, and, thus, the higher is the likelihood that the next unobserved rank will be similar to the known rank. Take a look a the Table 3 with the calculated prior likelihood for the unique rankings. We also added a column with probabilities considering the rankings as one class.

Table 3. Comparison of prior probability and prior similarity

y			$P(y)$	$L(y)$
A	B	C	0.500	0.708
B	C	A	0.333	0.583
A	C	B	0.167	0.792

As stated above, the ranking $\{A, C, B\}$, due to its closed similarity to the other two rankings, results in a higher level of likelihood. It is worth to note the since we measure the likelihood as the distance between ranks, it should not add to one as in case of probability.

The similarity of rankings based on the attribute values given an instance x_i (equivalent to the conditional probability in classification) is:

$$L(x_{i,a}|y) = \frac{\sum_{i:x_{i,a}=v_{a,j}} \rho(y, y_i)}{|\{i : x_{i,a} = v_{a,j}\}|} \tag{7}$$

Table 4 demonstrates the logic behind the conditional ranking similarities. Notice that ranking $\{A, C, B\}$ does not have a corresponded attribute and thus there is a probability of 0. However, the similarity approach ensures that there will be no zero similarities as there is always distance between the rankings.

Table 4. Comparison of conditional probabilities and similarities based on attribute Sunny

y			$P(Outlook = Sunny\|y)$	$L(Outlook = Sunny\|y)$
A	B	C	0.33	0.500
B	C	A	1.00	0.750
A	C	B	0.00	0.750

Applying equation (5), we get the estimated likelihood of

ranking y:

$$L(y|x_i) = \frac{\sum_{i=1}^{n} \rho(y, y_i)}{n} \left[\prod_{i=1}^{n} \frac{\sum_{i:x_{i,a}=v_{a,j}} \rho(y, y_i)}{|\{i : x_{i,a} = v_{a,j}\}|} \right] \quad (8)$$

and the similarity-based adaptation of naive Bayes for label ranking will output the ranking with the higher $L(y|x_i)$ value:

$$\hat{y} = \arg\max_{y} L(y|x_i)$$

6 Experiment Results

We empirically tested the proposed adaptation of the Naive Bayes algorithm for learning label rankings on some ranking problems obtained from metalearning applications. In these metalearning datasets, each example (x_i, y_i) represents a machine learning problem, referred to here as base-level dataset (BLD). The x_i is the set of metafeatures that represent characteristics of the BLD (e.g., mutual information between symbolic attributes and the target) and the y_i is the target ranking, representing the relative performance of a set of learning algorithms on the corresponding BLD. More details can be found in [1]. The code for all the examples in this paper has been written in R ([10]).

We used the following metalearning datasets in our experiments:

- `class`: these data represent the performance of ten algorithms on a set of 57 classification BLD. The BLD are characterized by a set of metafeatures which obtained good results with the KNN algorithm [1].
- `regr`: these data represent the performance of nine algorithms on a set of 42 regression BLD. The set of metafeatures used here has also obtained good results previously [9].
- `svm-*`: we have tried 5 different metalearning datasets describing the performance of different variants of the Support Vector Machines algorithm on the same 42 regression BLD as in the previous set. The difference between the first three sets, `svm-5`, `svm-11` and `svm-21` is in the number of different values of the kernel parameter that were considered. In the remaining two datasets, `svm-11-0.001` and `svm-11-0.128`, the number of kernel parameter values is 11 but the value of the ϵ parameter is varied. The metafeatures used measure properties of the kernel matrix [8].

Given that the attributes in the metalearning datasets are numerical and the NB algorithm is for symbolic attributes, they must be discretized. We used a simple equal-width binning method using 10 bins. The baseline is a simple method based on the mean rank of each algorithm (or parameter setting) over all training BLDs [1].

The performance of the label ranking methods was estimated using a methodology that has been used previously for this purpose [1]. It is based on the leave-one-out performance estimation method because of the small size of the datasets. The accuracy of the rankings predicted by methods was evaluated by comparing them to the target rankings (i.e., the rankings based on the observed performance of the algorithms) using the Spearman's correlation coefficient (Eq.1).

Table 5. Experimental results of the adapted Naive Bayes algorithm for label ranking compared to the baseline.

Dataset	NBr	Baseline	p-values
class.7	0.506	0.479	0.000*
regr.sel.num	0.658	0.523	0.056
svm5.ker	0.326	0.083	0.000*
svm11.ker	0.370	0.144	0.029*
svm21.ker	0.362	0.229	0.055
svm.eps01.ker	0.369	0.244	0.091
svm.eps0128.ker	0.372	0.251	0.268

The results are presented in Table 5[3]. As we can see, the adapted naive Bayes for label ranking algorithm outperformed the baseline in all examples.

7 Conclusion

In this paper we presented an adaptation of the naive Bayes algorithm for label ranking which is based on similarities of the rankings. We established a parallel between the similarity of the rankings and the likelihood considering the distance between the rankings. We tested our new algorithm on a number of metalearning datasets and conclude that it constantly outperforms the baseline. We will continue our work on improving the algorithm to produce more statistically significant results.

REFERENCES

[1] PB Brazdil, C Soares, and JP Costa, 'Ranking learning algorithms: Using ibl and meta-learning on accuracy and time results', *Machine Learning*, **50**(3), 251–277, (2003).

[2] W Cheng, J Hühn, and E Hüllermeier, 'Decision tree and instance-based learning for label ranking', pp. 161–168, (2009).

[3] O Dekel, C Manning, and Y Singer, 'Log-linear models for label ranking', *Advances in Neural Information Processing Systems*, **16**, (2003).

[4] S Har-Peled, D Roth, and D Zimak, 'Constraint classification: A new approach to multiclass classification', volume 51, p. 365, (2002).

[5] E Hüllermeier, J Fürnkranz, W Cheng, and K Brinker, 'Label ranking by learning pairwise preferences', *Artificial Intelligence*, **172**(2008), 1897–1916, (2008).

[6] G Lebanon and JD Lafferty, 'Cranking: Combining rankings using conditional probability models on permutations', p. 370, (2002).

[7] TM Mitchell, *Machine Learning*, 1997.

[8] C Soares and P Brazdil, 'Selecting parameters of svm using meta-learning and kernel matrix-based meta-features', *Proceedings of the 2006 ACM symposium on Applied computing*, (Jan 2006).

[9] Carlos Soares, *Learning Rankings of Learning Algorithms*, Ph.D. dissertation, 2004.

[10] R Development Core Team, *R: A Language and Environment for Statistical Computing*, 2008. ISBN 3-900051-07-0.

[11] L Todorovski, H Blockeel, and S Dzeroski, 'Ranking with predictive clustering trees', *Lecture notes in computer science*, 444–455, (2002).

[12] S Vembu and T Gärtner, 'Label ranking algorithms: A survey', *Preference Learning. Springer*, (Jan 2009).

[13] M Vogt, JW Godden, and J Bajorath, 'Bayesian interpretation of a distance function for navigating high-dimensional descriptor spaces.', *Journal of chemical information and modeling*, **47**(1), 39, (2007).

[3] (*) denotes statistically significance at 5% confidence level

Recurring Concepts and Meta-learners

João Gama[1] and **Petr Kosina**[2]

Abstract

This work addresses data stream mining from dynamic environments where the distribution underlying the observations may change over time. In these contexts, learning algorithms must be equipped with change detection mechanisms. Several methods have been proposed able to detect and react to concept drift. When a drift is signaled, most of the approaches use a forgetting mechanism, by releasing the current model, and start learning a new decision model. Nevertheless, it is not rare for the concepts from history to reappear, for example seasonal changes. In this work we present a method that memorizes learnt decision models whenever a concept drift is signaled. The system uses meta-learning techniques that characterize the domain of applicability of previous learnt models. The meta-learner can detect re-occurrence of contexts and take pro-active actions by activating previous learnt models. The main benefit of this approach is that the proposed meta-learner is capable of selecting similar historical concept, if there is one, without the knowledge of true classes of examples.

Keywords

Data Streams, Concept Drift, Meta-learners, Recurrent Concepts

1 Motivation

We are living in the time of information. Information becomes more and more valuable for any kind of business. It can help to make strategic decisions, produce better products, find new customers... But it can also save lives when used in medicine, pharmacy etc. Therefore, with the increasing computational power and storage space, vast amounts of data is being produced and gathered every moment. It is not possible for human to manually study all the data and look for the interesting or valuable pieces of information. Once again there are computers to aid us to automatically process data, search for and retrieve relevant answers. But not all of the data posses the same characteristics so we need to study and develop new techniques that would best fit to particular problems.

Some of the sources can produce more or less static data, but more often we are faced to evolving problems, therefore also methods for data analysis should not be stationary, but should be able to adapt.

Observing the behaviour of nature can give us more than just the idea of evolution of things. There is also the tendency of reappearence. The most obvious example could be seasonal change and the reactions of animals or people to those changes. Even long thime ago people knew that after summer there comes autumn and then winter etc. and prepared reserves for worse conditions. That is why even computer decisions should consider using historical information and try to predict its recurrence.

In this work we present an approach that possesses both qualities, adaptation and using historical information. But the novelty and main motivation lies somewhere else. Imagine a situation in a bank where there are records of their clients. Computer systems can help decide if bank should give a loan to certain client. However more new clients will come and ask for loan before the bank can label clients that were reliable or not. Computer systems predict labels for records, but usually are not able to use entries without their labels for the evaluation. Since the economic situation can change, the predictions will not correspond to current state anymore and it will take some time to make corrections in decision making. We were interested if we could use the records without labels and the recurrence of history to react faster on changes.

2 Related Work

Existing works in mining data streams usually deals with concept drifts by detecting them and learning a new classifier that is used either alone as in [2] or as a part of ensemble like in [12, 14]. But there are not many of them taking into consideration the recurrence of concepts in the streams.

An ensemble classifier with recurrence taken into account is presented in [8]. In their work, classifiers of new concepts are stored in global set. Only the models with performance on preceding data points better than threshold are part of ensemble for labelling the examples. The threshold is set as selected fraction of error of classifier that predicts randomly (e.g. probability of classification to class c is equal to c's class distribution). This ensures that relevant classifiers, with weight assigned, are in the ensemble. If none of the classifiers in global set or the ensemble of them perform better than permitted error, a new classifier is built. Classifiers are tested, selected to ensemble or new one is created with every labelled chunk of examples.

Also in [16] a method reusing historical concepts is proposed. It uses proactive approach, e.g selects concept from history that will most likely follow after the current concept according to transition matrix. History of concepts is treated like a Markov Chain with concepts as states. If this approach does not have enough information to distinguish among some historical concepts, it makes decision based on historical-reactive scheme, which means it tests the historical concepts with recent examples and the one with the best accuracy is chosen. In case none of this approaches has satisfying accuracy, a new concept is learnt. Building a concept history can be boosted measuring conceptual equivalence (CE). CE compares classifications of classifier from new concept to results of each of the historical ones and computes score for each pair. If the score for one of the historical concepts is above defined threshold, new concept replaces old one in

[1] LIAAD-INESC Porto, FEP-University of Porto email: jgama@fep.up.pt
[2] LIAAD-INESC Porto, FI-Masaryk University, Brno email: petr.kosina@inescporto.pt

concept history.

In [4] Conceptual Clustering and Prediction (CPP) is presented to handle streaming data with recurrent concepts. Short batches of examples are transformed into conceptual vectors describing them (e.g. mean and standard deviation for numeric atributes, probability of attribute given the class for nominal). Conceptual vectors are clustered by their distance and for each cluster a classifier is learnt.

Recurrent concepts are also considered in [6], where multiple windows are used for tracking the concept drift. By predicting the rate of change, size of window could be adapted and when drift is estimated, repository of stored historical concepts is checked for recurrence. Concepts are described by averages of attributes (numeric) and similarity is measured by any feature distance metric.

In this work we present another approach to select older models learnt on data with similar underlying concept. Single classifier is used for predicting class labels and meta-learner is used to choose the most appropriate model from history. Every time a new classifier is not sufficient and warning is signaled, referees are asked for predictions. Whenever there is drift detected the classifier and its referee are stored in a pool for further use. Model is re-used when the percentage of votes of its referee exceeds some given threshold, otherwise new classifier is learnt. The idea of using such referees is taken from [9] where meta-learning scheme was used in off-line learning to select predictions from an ensemble of classifiers. We tried to apply the scheme in on-line learning with a single classifier for simplicity.

3 The Two-Layers Learning System

The system we propose uses a two layer learning scheme. Each layer trains its own classifier and receives its own data. The first layer receives the data stream and trains a classifier using the labeled examples. For each incoming example **x**, y, the current classifier predicts a class label. If the example is not classified, i.e. y = ?, the current model classifies the example, and reads the next example. Otherwise, e.g. the example is classified; we can compute the loss, and update the current decision model with the example. Assuming the 0-1 loss function, the prediction is either correct or incorrect, and an example is generated to train the second layer classifier. This example has the same attribute values as in the layer 0, but the class value is + if the example was correctly classified or if the example was misclassified. Doing so, the meta-learner learns the regions of the instance space where the base classifier performs well.

3.1 Base Model: Naive Bayes

Our approach does not depend on any certain learning algorithm. Any classifier could be used for either level 0 or level 1. Naive Bayes classifier was chosen for our experiments for good reasons. It is easy to implement and it is incremental. It has clear semantics, it is simple in representing, using and learning probabilistic knowledge, works well with continuous attributes and finally in [9] it is reported to have good results as meta-learner.

Prior probabilities, used in computation, are usually small numbers and the product of such numbers gives us even smaller numbers, which can easily lead to underflow while using computers. It is better to use the sum of logarithms and receive a score that is proportional to its probability. The formula is $\log P(c_i|\vec{x}) \propto \log P(c_i) + \sum_{k=1}^{n} \log P(x_k|c_i)$ For two class problem we can also use the expression in the form:

$$\log \frac{P(c_+|\vec{x})}{P(c_-|\vec{x})} \propto \log \frac{P(c_+)}{P(c_-)} + \sum_{k=1}^{n} \log \frac{P(x_k|c_+)}{P(x_k|c_-)} \qquad (1)$$

The class of the example is then assigned: $if \log \frac{P(c_+|\vec{x})}{P(c_-|\vec{x})} > 0$ class is $+$, $-$ otherwise.

Since we are dealing with data streams, it was necessary to use the incremental version of Naive Bayes, which needs to process each example just once. We compute the incremental version of Naive Bayes described in [3].

For handeling the continuous attributes, the classical method that approximates some distribution was chosen. In this case it was the normal or Gaussian distribution. We assume that $P(x_k|c_i) = \frac{1}{\sigma_{ki}\sqrt{2\pi}} exp\left(\frac{-(x-\mu_{ki})^2}{2\sigma_{ki}}\right)$ where μ_{ki} is mean and σ_{ki} is standard deviation.

3.2 Drift Detection

In many real-world situations we can see that behaviour of observed subject is changing over time. Such changes in class distributions are called drifts and there are several known methods to detect them.

We used approach from [2] which assumes that if the distribution of the examples is stationary, the error-rate of the learning algorithm will decrease when the number of examples increases. The drift detection method manages two registers during the training of the learning algorithm, p_{min} and s_{min} , where p is error-rate and s is standard deviation. Every time a new example i is processed those values are updated when $p_i + s_i$ is lower than $p_{min} + s_{min}$. We use a warning level to define the optimal size of the context window. The context window will contain the old examples that are on the new context and a minimal number of examples on the old context. In the experiments the warning level is reached if $p_i + s_i \geq p_{min} + 2s_{min}$ and the drift level is reached if $p_i + s_i \geq p_{min} + 3s_{min}$. Suppose a sequence of examples where the error of the actual model increases reaching the warning level at example k_w, and the drift level at example k_d. A new decision model is induced using the examples starting in k_w till k_d. It is possible to observe an increase of the error reaching the warning level, followed by a decrease. We assume that such situations correspond to a false alarm, without changing the context.

3.3 Learning with Model Aplicability Induction

When dealing with possibly infinite data streams one can expect there would be concept changes in data and the concepts could re-appear. We need to have some tool for recognizing if older model is appropriate for new data. Such a tool can be a referee which is a meta-learner working in similar fashion as in [9]. There are two layers in our suggested approach. The first layer is the primary model that serves as normal classifier (or level 0 classifier). Then the second layer denoted as referee (or level 1 classifier).

As in real situations there is delay between obtaining example and observing true label. Level 0 classifier makes the prediction when the example arrives. Once the true value of example's class is observed new prediction for the example is made so that the evaluation would reflect current state of the model. Then the classificator is updated and the example is passed, if defined conditions for learning are met, to the referee with a new class atrribute, which reflects whether or not the prediction was correct. The data set for referee is therefore created from all the example attributes, apart from the class attribute,

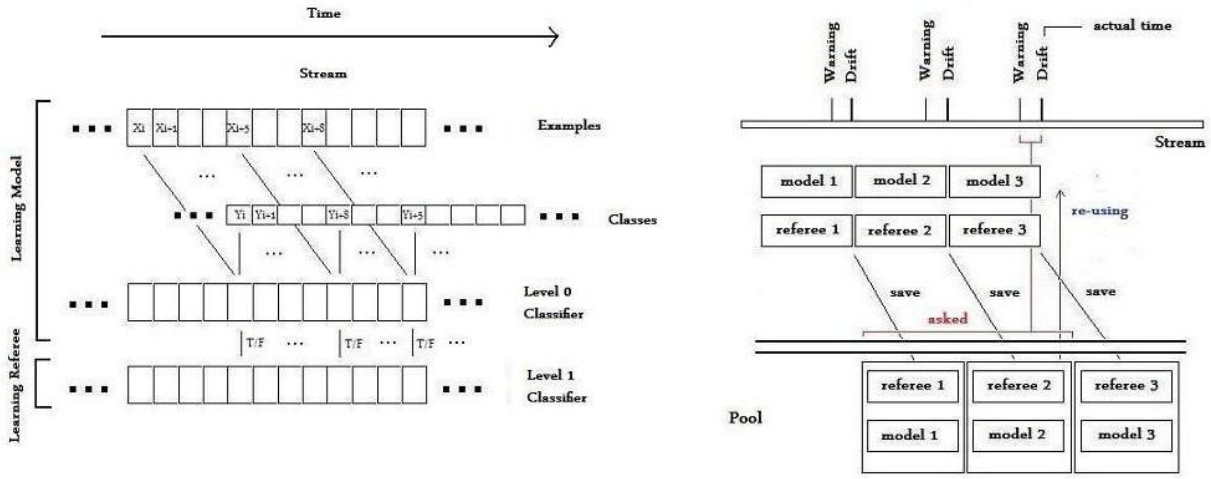


Figure 1. (left) Learning process. Once obtained the true label is passed along with the example attributes to level 0 classifier. Then example with true/false label is used for training level 1 classifier.
(right) Referee strategy is shown in the time of third drift. After warning, the referees in the pool were asked, model 2 was selected for re-using while model 3 and its referee were stored in the pool.

which is replaced by 0 (or false) when the prediction was wrong or 1 (or true) when the prediction was correct. The process can be illustrated by picture 1 (left).

Referee is built in the same time when primary model is built. First examples are not used in referee learning, because the misclassifications of the model are mainly due to the lack of information (in our case it was 50 examples). In fact, we have two parallel classificators on one data stream.

When there is a warning we store the incoming examples in a short term memory. This has two reasons. First, if the warning was a false alarm then we use those examples for learning like there was no warning at all. Second, if there is a drift, we use these recent examples, potentially examples from new concept, for building a new model. At this time, new referee is not learning for the same reason like in the beginning.

Part of the reaction on warning is that we stop learning primary model and we start asking the referees about the performance of their models. If model is still in the warning phase after defined number of examples, the probability of false alarm is much smaller and the examples in memory that still did not recieve their labels are also used to ask referees. If referee's prediction about the applicability of its model is greater than selected threshold, the correspondent model is chosen to be used for evaluating next examples. Otherwise process continues till drift level defined by drift detection is reached or referee's threshold value is exceeded, which is checked every 100 examples.

This is the main advantage of using referees. Many examples that do not have their labels are stored in the memory and we could exploit them to predict the change sooner with reusing old classifiers. That way it is not necessary to pasively wait till drift is reached, but proactively select recurrent model.

All new models and their referees are stored in a pool, like on the picture 1 (right). When model is to be re-used, it is taken from the the pool together with its referee and updated version is put back after drift occurred. For similar concepts there is one classifier with one referee that are continuously updated.

3.4 Loading Data

Data for stream mining can come and be processed either one by one or in batches. Both situations can be seen in applications. Although waiting for certain amount of examples and then processing the whole batch could be more efficient, data points in this work are taken one by one. Every time a new example is loaded, it is passed to the primary classificator to make a prediction. To simulate behaviour in real world, the true value of the class attribute needed for evaluation of model is not observed immediately, but after some delay (500 examples in this situation). Meanwhile the examples and predictions are stored in the memory.

3.5 Optimizing Performance

While making experiments with our referee-advised recurrence tracking method we encountered several problems closely connected to skewness of data. The distribution of examples for referees reflects the error-rate of level 0 classifier. Good performance of the primary classifier means that there will be a lot of positive examples and much less negative examples. This implies the need for some method to deal with this skewed data.

A technique without parameters is used to eliminate skewness. The referee makes decision about classifier's prediction and learn it just when it's decision is wrong e.g. referee predicted the classifier to be correct on incoming example and classifier made a mistake or vice versa. In order to be able to make decisions the referee learns from some examples in the beginning, it was 100 examples in our experiments.

In our case, we chose Naive Bayes classificator to be the referee. Since this meta-learning is a two class problem, we implemented the classificator as log-odds version of Naive Bayes defined by equation 1. The decision of classifier is chosen to be in favor for $true$ only if the log-odds score is above 0.1 not 0 like in usual cases. This scenario is more pesimistic one because it is better to start building a new model than using an old inappropriate one.

Another constraint was introduced in the process of saving referees. Since referees with practically the same means and standard de-

viations for both classes ($true$ and $false$) were obtained, we decided to save them only in certain conditions (reaching minimum quality): $if\left(\sum|\mu_i^1-\mu_i^2|\right)>\phi$, where ϕ is selected threshold (in this case 0.002) and μ_i^1,μ_i^2 are means of attributes of $true$ and $false$ class respectively.

Bad quality and skewness of data, could lead to a situation, where referee predicts model to be correct in 100 % cases because of higher prior probability of positive class, even though above mentioned precausions were taken. Therefore results with 100 % are not taken into account.

3.6 Data Sets

SEA Concepts. In order to test the method we proposed, we needed data that changes over time e.g. concept drifts are present. We used artificial data described in [12]. Samples of data concepts with the relevant features are plotted in the figure 2(A). Each concept had 15,000 examples with about 10 % of class noise.

Description in [12] holds for our data set with one improvement. Since we wanted to re-use the old models and to study how our method works, we doubled the size by concatenating original set with another copy of it. We obtained data set with 120,000 examples and eight concepts such that concept 1 is the same like concept 5, concept 2 equals concept 6 etc. Figure 2(B) illustrates the data layout.

This way the recurrence is present and concepts are not just similar, but exactly the same. Normally we would probably never encounter such strong recurrence, but it serves well for studying the problem.

LED Data. Drift data stream generator from [5] with default settings was used to create another data set. Every 10,000 we change the number of attributes with drift. We generated 120,000 examples with the following randomly chosen numbers of attributes with drift: $2, 4, 3, 1, 4, 3, 2, 5, 1, 4, 3, 5$.

Intrusion. This data set was used in KDD Cup 1999 Competition described in [17]. The full dataset had about five million connection records, but we also used data set with only 10 % of the size finding it more illustrative.

The original task had 24 training attack types. The original labels of attack types were changed to label *abnormal* keeping the label *normal* for normal connection. This way we simplified the set to 2 class problem.

3.7 Using the Referee vs Simple Drift Detection

Handeling concept drifts is well-studied problem in data stream mining and many solutions could be found and used. For example by detecting the drift and building a new model while forgetting the old one, we can achieve better results. Furthermore, we can store examples in short-term memory during the warning phase and use these recent examples, potentionally representants of a new concept, for learning a new model.

In this paper we introduced a new way to improve these techniques. The idea is to store the old models instead of forgetting them and try to re-use them when there is similar underlying concept in new data.

The threshold for re-using old model was set to 70 % in all data sets. Different settings for different sets could improve the performance, but we would like to use more universal threshold. To avoid unnecessary testing when there is false alarm the method waits and if it is still in warning phase after 350 examples all unlabeled examples from the memory are tested by referees.

On the figures 3,4 (left) there are results for SEA Concept data set. As we can see re-using model from concept 1 for concept 3 lead to slightly worse results, because those two are not that similar as concept 2 and concept 4 (see figure 2), where re-using performed the same as building a new model. Moreover, re-using model from concept 2, learnt also on concept 4, on data from the sixth one lead to better results than in the case of learning new classifier. In concept 7 we can see the same situation as in concept 3.

The goal of predicting drift and recurrence was quite succesful, three out of four cases of re-using reported drift sooner. 183.5 points improvement on average considering times when drift occured. Average number of examples in warning, considering all the warning phases, decreased by 80 examples, which was improvement of 9.25 %. When model was re-used, it was immediately after those 350 examples. It would suggest lowering the number and drift could be reported even sooner, but with less examples in warning referees tended to report drift and change the model when there were false alarms.

We were interested what would be the performance if, in the case of SEA Concepts data, the referee would choose exactly the models that were learnt on the same concepts in history. It should be mentioned that this experiment was conducted just because the set was artificial and the concepts and recurrence was known. In real data sets this kind of learning is impossible to do so. We manually reused the historical concepts e.g. first model for fifth concept etc. On figure 4 (right) we can see that the performance also was not ideal. Slow continual increase in error-rate caused the warning to be reported much sooner and second model learnt from examples of fifth concept that were stored in short memory during the warning phase.

The figure 5 (left) shows the results of using proposed method on LED data. Both approaches (referee-adviced and without re-using models) are same till the concept 11, where referee choses model 6. As can be seen in data description, both concepts were generated with same number of attributes with drift, therefore the performance was better than without re-using any model. However, the model had increading tendency in error rate and warning was reported very soon. The situation was similar to the one in experiment with true models in SEA Concepts. In this case new incorrect drift was detected, model 10 was re-used which caused decrease in accuracy. The actual drift after concept 10 was detected 1022 examples sooner, but the other true drift was then delayed 301 examples.

Running our method on data from KDD Cup gave us results captured on figure 5 (right). The set tend to have structure of many different-lenght groups of consecutive instances of the same class. Drifts occured in the data because different types of attack, and therefore different protocols etc. used for the connection, were labeled with the same class, but also because normal connections had different properties. Those concepts were re-appering in the data set making it suitable for the tests.

Even though in most of the cases drifts were abrubt and quickly detected, some minor improvements were to be found.

3.8 Using Model Advice vs Referee Advice

In this secton we compare results obtained by our approach and those obtained by testing the accuracy of models during warning phase instead of asking referees. For re-using selection was the threshold set to approximately average performance of the models. We have to say it was not expected that referees would have better results. It is obvious that with the information about true label it is much easier to select proper model, but labels in real situations are obtained

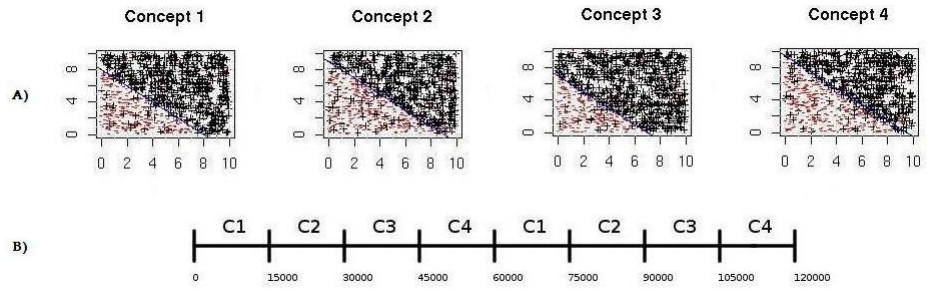


Figure 2. A) samples of data concepts with 1000 examples each and B) layout of concepts in experimental data

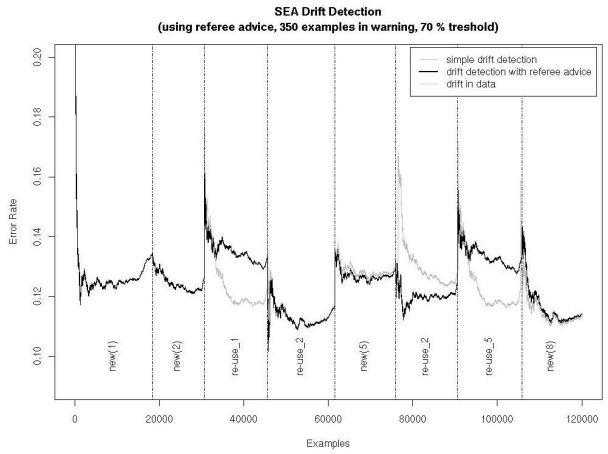


Figure 3. Comparison of error-rates with re-using models (70 % threshold) and without re-using on SEA Concepts

after delay while attributes of examples are available. Therefore this section shows how close or far are referee's results.

As in re-using true models in concept 5 there was warning reported too early. But this time re-using of second model was not forced so new model was learnt from warning examples combining examples from fifth and sixth concept and therefore drifted very soon. Too early warning occured also at the end of concept 7. Otherwise model-advised approach was prevailing as expected. Comparison of these two approaches is illustrated on the figure 6.

Model-advised classifier also performed better on LED data (figure 7 on the left), as expected, with the exception in seventh concept, where there was data generated with same settings as in re-used concept 1. Nevertheless, re-using caused that non-existing drift was reported and new classifier was learnt in the middle of actual concept in data.

Model-advised approach on Intrusion data set 7 (right) was not very succesful. Despite of setting the threshold to 99.5 % drifts were reported too often. It resulted in needless changes of context and therefore downgrading the overall performance. Compare 64 drifts of simple drift detection to 372 of model-advised approach.

4 Conclusions

In this work new method for tracking recurrent concepts was introduced. Even though there was not great improvement on overall accuracy, the method showed it could detect drift sooner with re-using older models. There are is still a lot of space for improvement and future. Other types of classifiers or ensemble of classifiers could be used and perform better either as primary classsifier or as referee. We believe that future further study of this method could bring interesting results.

REFERENCES

[1] Gama, João and Fernandes, Ricardo and Rocha, Ricardo. Decision Trees for Mining Data Streams. Intelligent Data Analysis, IOS Press, Pages 23-45, 2006.

[2] Gama João, Medas Pedro, Gladys Castillo, and Rodrigues Pedro. Learning with Drift Detection. *Proc. of the 17th Brazilian Symposium on Artificial Intelligence (SBIA 2004)*, Volume 3171, page 286–295 - October 2004.

[3] Gama, João, Gaber, Mohamed Medhat (Eds.). Learning from Data Streams: Processing Techniques in Sensor Networks. Springer, 2007.

[4] Katakis, Ioannis, Tsoumakas, Grigorios, Vlahavas, Ioannis. An Ensemble of Classifiers for coping with Reurring Contexts in Data Streams. *18th European Conference on Artificial Intelligence*, IOS Press, Patras, Greece, 2008.

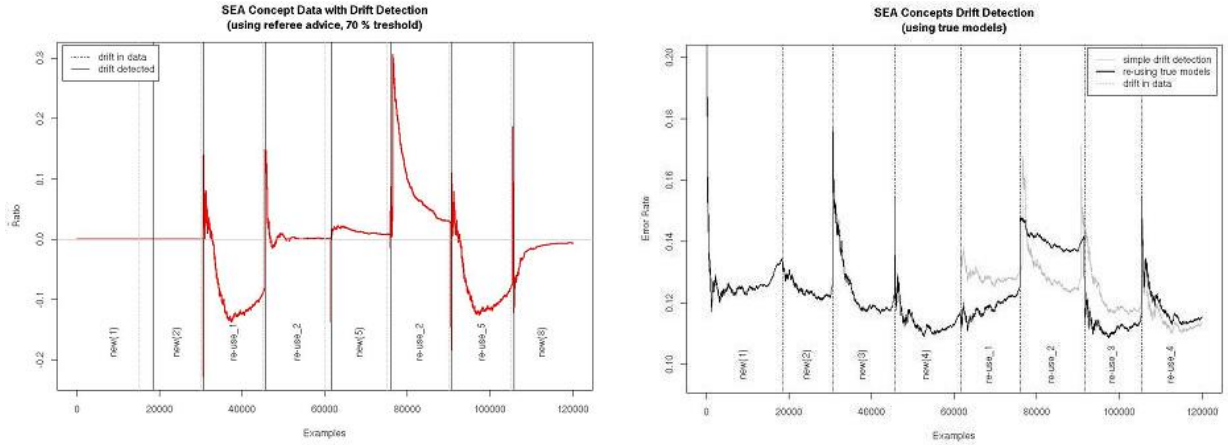


Figure 4. (left) Logarithm of ratio between error-rates without and with re-using models on SEA Concepts. Curve above 0 is in favor of method with referees. (right) Comparison of error-rates with re-using true models and without re-using on SEA Concepts.

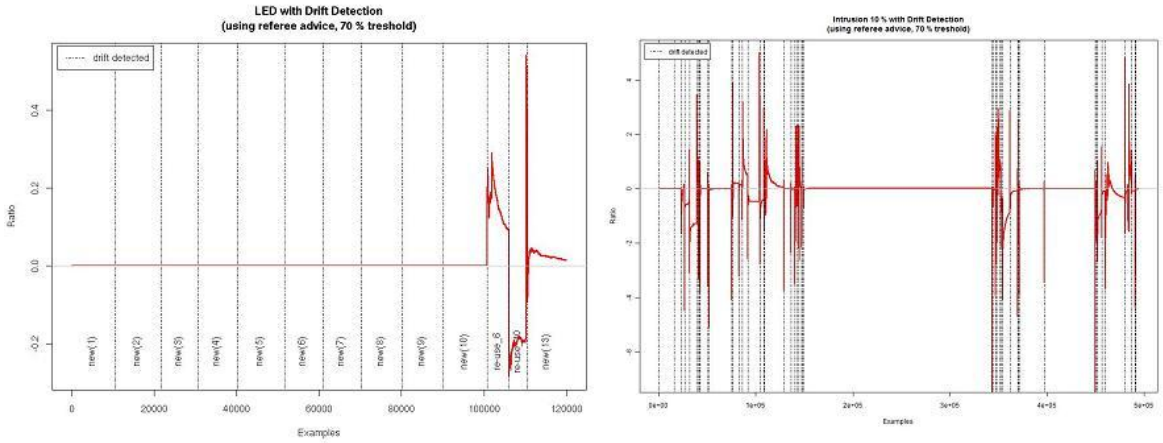


Figure 5. Ratio between error-rates without and with re-using models on LED data set (left) and Intrusion (right). Above 0 is in favor of referees

[5] Kirkby Richard. Massive Online Analysis. University of Waikato, Hamilton, New Zealand, 2007. http://sourceforge.net/projects/moa-datastream/

[6] Lazarescu, Mihai. A Multi-resolution Learning Approach to Tracking Concept Drift and Recurrent Concepts. PRIS, 2005.

[7] Mohamed Medhat Gaber, João Gama, Auroop R. Ganguly, Olufemi A. Omitaomu, Ranga Raju Vatsavai. Knowledge Discovery from Sensor Data. Taylor Francis, 2008.

[8] Sasthakumar Ramamurthy, Raj Bhatnagar. Tracking Recurrent Concept Drift in Streaming Data Using Ensemble Classifiers. *Proc. of the Sixth International Conference on Machine Learning and Applications*, Pages 404-409, 2007.

[9] Alexander K. Seewald and Johannes Fürnkranz. Grading Classifiers. Austrian Research Institute for Artificial Intelligence, 2001, OEFAI-TR-2001-01, Wien, Austria.

[10] Jeffrey C. Schlimmer and Richard H. Granger Jr. Incremental Learning from Noisy Data. Machine Learning 1: 317-354 1986.

[11] Stanley K.O., Learning concept drift with a committee of decision trees, Tech. Report UT-AI-TR-03-302, Department of Computer Sciences, University of Texas at Austin, USA, 2003.

[12] W.Nick Street and YongSeog Kim. A streaming ensemble algorithm SEA for large-scale classification. In *Proc. seventh ACM SIGKDD international conference on Knowledge discovery and data mining*,pages 377-382. ACM Press, 2001.

[13] Tsymbal, Alexey. The Problem of Concept Drift: Definitions and Related Work. http://citeseer.ist.psu.edu/tsymbal04problem.html

[14] Wang, H., Fan, W., Yu, P., Han, J.: Mining concept-drifting data streams using ensemble classifiers. In: *Proc. KDD. (2003)*

[15] Witten, Ian H. and Frank, Eibe. Data Mining: Practical Machine Learning Tools and Techniques With Java Implementations. Morgan Kaufmann Publishers, 1999.

[16] Ying Yang, Xindong Wu, and Xingquan Zhu. Mining in Anticipation for Concept Change: Proactive-Reactive Prediction in Data Streams. In *the Proc. 11th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2005)*, pages 710-715.

[17] http://kdd.ics.uci.edu/databases/kddcup99/task.html

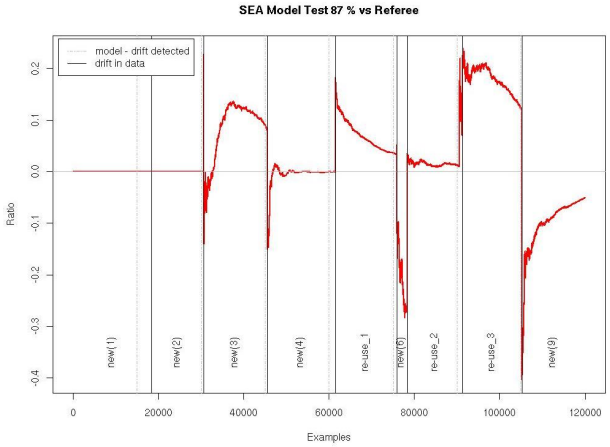


Figure 6. Ratio between error-rates with model-advised and referee-advised re-using on SEA Concepts - above 0 is in favor of model-advised approach

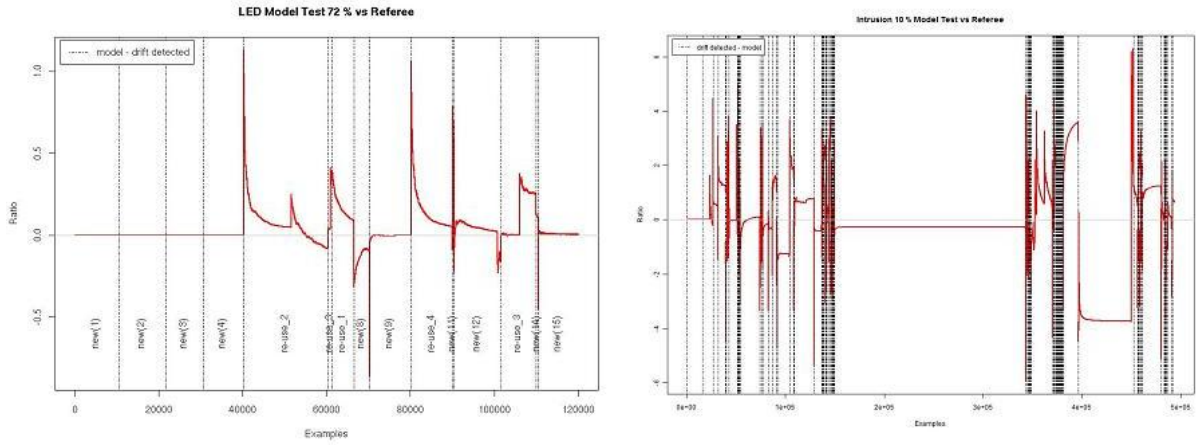


Figure 7. Ratio between error-rates with model-advised and referee-advised re-using on LED data (left) and on Intrusion (right) - above 0 is in favor of model-advised approach

Author Index